

(Hello World)

WELCOME TO THE MAGAZINE

NEW!

JOIN THE CLUB

CODE CLUB CO-FOUNDER
CLARE SUTCLIFFE SHARES HER
THOUGHTS ON BREAKING THE
5,000 UK CLUBS MILESTONE,
WORLDWIDE EXPANSION,
AND WHAT 2017 HAS IN STORE



Issue 1

Spring Term 2017 helloworld.cc

FOR COMPUTING & DIGITAL MAKING EDUCATORS

PAPERT'S LEGACY

What we can learn from
the father of Logo, and
constructionism's role today

LEARN NEW COMPSCI SKILLS

Start plugging knowledge gaps today

PROJECT QUANTUM

Do your pupils really understand? Quantum
aims to help you ask the perfect question

LESSON PLANS AND TUTORIALS

Ideas to inspire and help you and your students

CREATE A MAKERSPACE

How to introduce a digital
making space in your school

PLUS

IS COMPUTING BECOMING MORE EXCLUSIVE? • ITALY PUTS MAKERSPACES IN EVERY SCHOOL • GRADUATE FROM SCRATCH TO PYTHON • CODING INTERACTIVE FICTION • PIONEERS PROGRAMME LAUNCHES • YOUR DIGITAL MAKING QUESTIONS ANSWERED • 3D GRAPHICS WITH FREE SOFTWARE • INSIDE PICADEMY USA • THE IMPORTANCE OF PLAYFUL COMPUTING



/ UNITED KINGDOM



Start a Code Club in Your School!

Code Club is a nationwide network of volunteers and educators who run free coding clubs for children aged 9-11.

Our aim is to inspire the next generation to get excited about computer science and digital making.



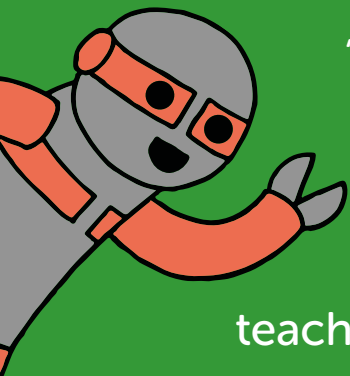
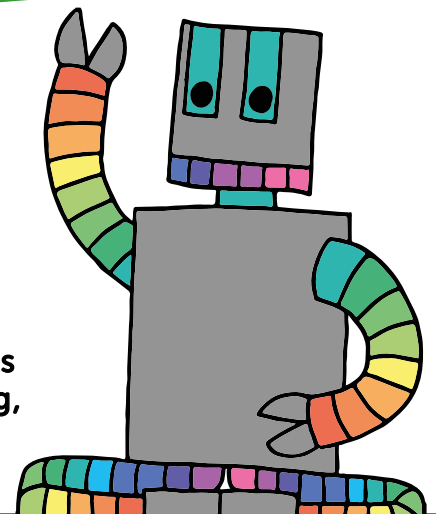
Code Club is free



Code Club is flexible



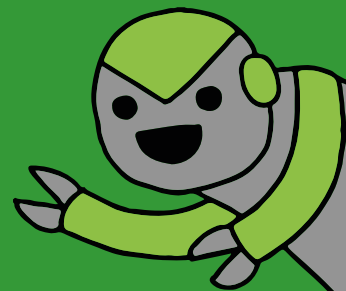
Code Club develops skills including logical thinking, creativity, and resilience



"Code Club has helped tremendously with the children's confidence and engagement in coding and computing"

Caroline Harding, Year 4 teacher

We have over 5,000 clubs across the UK, teaching more than 70,000 children to code - join us!



Find out more at: codeclub.org.uk



Code Club is part of the Raspberry Pi Foundation, Registered Charity Number 1129409

HELLO, WORLD!

Welcome to the zeroth edition of our new magazine for digital making and computing educators.

We've seen a shift in the culture around young people and technology, moving from a time when they were content with staying in touch and using other people's programs, to one where many are collaborating on digital projects across a whole range of media and technologies. Educators play a vital role: setting challenges, broadening horizons, and explaining ideas.

Hello World has been written for anybody who's introducing young people to computing and digital making, including primary and secondary teachers, volunteers, and parents. The magazine will be free for all, forever online, and free in print for teachers and educators based in the UK. Visit helloworld.cc to learn more.



Hello World is a collaborative project; it's a magazine by educators, for educators. Our writers are directly or indirectly involved in education, at and beyond school. We're eager to have an authorship as diverse as its readership, so get in touch if you'd like to write for us: miles@helloworld.cc.

It's a magazine that draws on the expertise and experience of the Raspberry Pi Foundation and Computing At School (CAS), part of BCS, the Chartered Institute for IT; it's the successor to CAS's Switched ON newsletter, edited by Roger Davies. His is a hard act to follow, and our hope is that Hello World will hold true to the vision behind Switched ON.

Please enjoy, be inspired, and help out.

Miles Berry
Contributing Editor



FEATURED THIS ISSUE



CARRIE ANNE PHILBIN
DIRECTOR OF EDUCATION,
THE RASPBERRY PI
FOUNDATION

Carrie Anne Philbin is Director of Education at The Raspberry Pi Foundation, a Python Software Foundation and Computing At School board member, author, and YouTuber.



MITCHEL RESNICK
PROFESSOR OF LEARNING
RESEARCH, MIT MEDIA LAB

Mitchel develops new technologies and activities to engage people (particularly children) in creative learning experiences. His goal: help everyone learn to think creatively, reason systematically, and work collaboratively.



MICHAEL KÖLLING
VICE-DEAN FOR EDUCATION
AND PROFESSOR OF CS,
KING'S COLLEGE LONDON

Michael Kölling is a Professor of Computer Science at King's College London. His research interests are in programming languages, software tools, computing education, and HCI.

(HW)

EDITORIAL

Managing Editor
Russell Barnes
russell@helloworld.cc

Contributing Editor
Miles Berry
miles@helloworld.cc

Sub Editors
Laura Clay, Lorna Lynch

DESIGN

Critical Media
criticalmedia.co.uk

Head of Design
Dougal Matthews

Designers
Lee Allen, Mike Kay

Illustrator:
Sam Alder

Cover photography:
Greg Annandale

CONTRIBUTORS

Jane Abrams, Rik Cross, Lucy Hattersley, Phil King, Oliver Quinlan, Laura Sach, Marc Scott & Rob Zwetsloot

Sponsored by



Hello World is a joint collaboration:



Raspberry Pi



COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE
Part of BCS, The Chartered Institute for IT

This magazine is printed on paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001.

Hello World is published by Raspberry Pi (Trading) Ltd., 30 Station Road, Cambridge, CB1 2JH. The publisher, editor, and contributors accept no responsibility in respect of any omissions or errors relating to skills, products or services referred to in the magazine. Except where otherwise noted, content in this magazine is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0).



(HW)

CONTENTS

COVER
FEATURE

PAPERT'S LEGACY

Honouring the creator of Logo and the godfather of the maker movement, who has left an indelible mark on CS education

24

94

PICADEMY USA

Raspberry Pi's free teacher training program takes a trip across the pond

88

ROBOTICS IN SCHOOLS

Teaching by making robots

86

HOW TO START A CODE CLUB

It's incredibly simple to set up a Code Club in your school. We have some tips to help you out

NEWS AND FEATURES

08 NEWS

Computing education news from around the world

16 PHYSICAL COMPUTING

Carrie Anne Philbin talks about learning through making

18 #INSIGHT

The science behind Papert's philosophy and more

22 THE WORLD IN A BOX

Using VR to support learning

23 CAESAR CIPHERS

Mark Thornbear shows us how to do simple cryptography

24 PAPERT'S LEGACY

How the creator of Logo paved the way for modern computing education

32 GREENFOOT AND TURTLE

Modern coding environment
Greenfoot owes a lot to Logo

36 CS EDUCATION FOR EVERYONE

Understanding how to be more inclusive with computing

38 HOW TO BECOME A VFX ARTIST

The skills needed to make graphics for the silver screen

39 LEARNING THROUGH
INTERACTIVE FICTION

Paul Powell tells us about his new experiment

40 PROJECT QUANTUM

Can Quantum test students better?

42 THE INFLUENCE OF LOGO

The history and legacy of Logo

43 BCS SCHOLARSHIP SCHEME

Get a BCS scholarship to learn how to teach

44 EXPERIMENTS IN COMPUTING

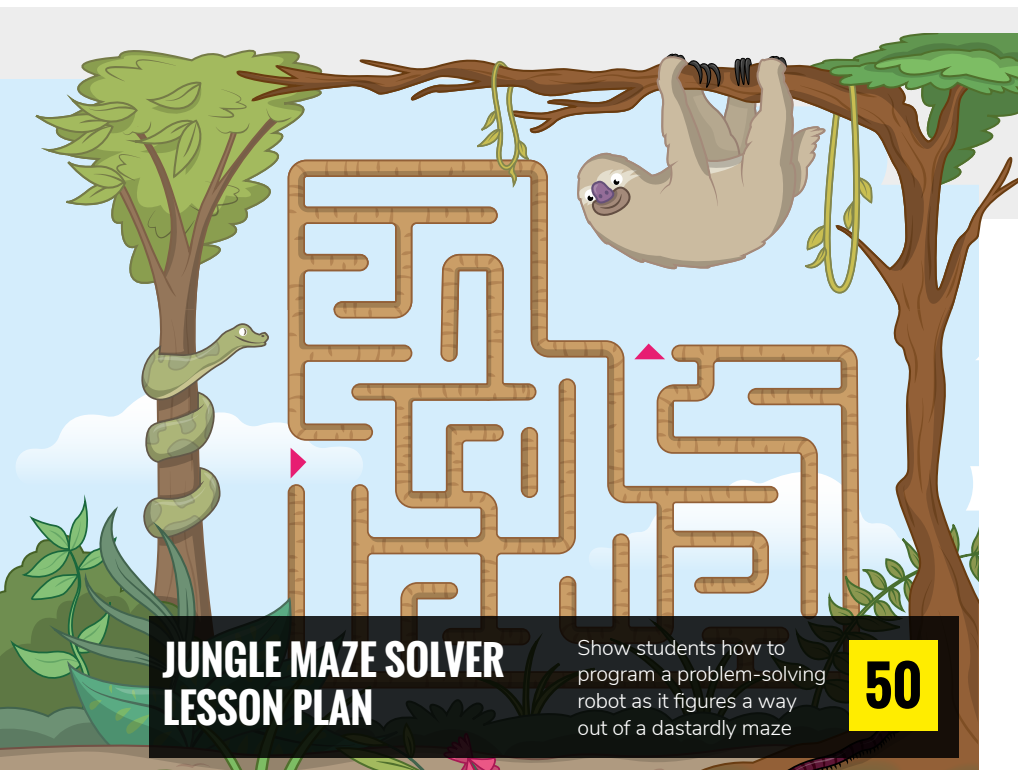
A report on a new way to teach CS

58 PLAYFUL COMPUTING

What to teach before programming

68 SET UP A MAKERSPACE

Ten steps to creating a makerspace



JUNGLE MAZE SOLVER LESSON PLAN

Show students how to program a problem-solving robot as it figures a way out of a dastardly maze

50



70

WHAT IS 3D PRINTING?

Everything you need to know to get by



16

PHYSICAL COMPUTING

Carrie Anne Philbin talks about how making is a catalyst for creativity

- 70 BLUFFER'S GUIDE: 3D PRINTERS**
What you need to know about 3D printers
- 74 FAQs**
Your CS questions answered
- 78 CODING APPS GROUP TEST**
Five great apps that can help teach programming
- 80 BOOK REVIEWS**
We review some helpful literature
- 82 BREAKING DOWN CODE**
Using maths to understand programming languages
- 84 CODE CLUB INTERVIEW**
Co-founder Clare Sutcliffe tells us Code Club's 2017 plans

- 86 SET UP A CODE CLUB**
Tips on how to start a Code Club
- 88 ROBOTS IN SCHOOLS**
CS education through robotics
- 92 CS FOR HIGH SCHOOL**
Code.org's Computer Science course is for all high-schoolers
- 93 EMBRACING FAILURE**
Nick Provenzano tells us about the impossible
- 94 PICADEMY USA**
A report on teacher training in America
- 98 VALUES AND ETHICS**
Thinking more about Computer Science

LEARNING

TUTORIALS & LESSON PLANS

- 46 MOVING FROM SCRATCH TO PYTHON**
Help students jump from visual to text-based programming languages
- 50 JUNGLE MAZE SOLVER**
Get your students programming a robot capable of solving a maze
- 52 ESCAPE FROM RAVENSWOOD MANOR**
Build and play a text-based adventure
- 54 MY AMAZING CASTLE**
Help students make a castle, and the dragon that lives in it, with Scratch
- 56 FLAGS OF THE WORLD**
Get a turtle robot to draw flags from around the world
- 60 TURTLE GRAPHICS IN SCRATCH**
Easy-to-adapt guides to using a turtle with Scratch
- 62 3D MODELLING WITH BLENDER**
How to use the free 3D modelling software in the classroom
- 64 SNAP! - BEYOND SCRATCH**
Like a more advanced Scratch, Snap! allows for progression up to 16- to 18-year-olds



“HELLO, WORLD!”

Everything you need to know about the new computing and digital making magazine for educators...

Q WHAT IS HELLO WORLD?

A Hello World Magazine is the new magazine for computing and digital making educators. Written by educators, for educators, the magazine is designed as a platform to help you find inspiration, share experiences, and learn from each other.

Q WHO MAKES HELLO WORLD?

A The magazine is a joint collaboration between its publisher Raspberry Pi and Computing at School, part of BCS, The Chartered Institute of IT. Hello World is sponsored by BT.

Q WHY DID WE MAKE IT?

A There's growing momentum behind the idea of putting computing and digital making at the heart of modern education, and we feel there's a need to do more to connect with and support educators inside and outside the classroom.

Q WHEN IS IT AVAILABLE?

A Your new 100-page magazine will be available three times per year in time for each new term in January, April, and September. Would you like it to be available more frequently? Let us know!



IT'S FREE!

Hello World is free now and forever as a Creative Commons PDF download. You can download every issue from helloworld.cc. Visit the site to see if you're entitled to get a free print edition, too.

GET INVOLVED TODAY

There are numerous ways you can get involved in the magazine.
Here are just a handful of ideas to get you started:

- **Give us feedback**

Help us make your magazine better - your feedback is greatly appreciated.

- **Ask us a question**

Do you have a question for our FAQ section or a bugbear you'd like to share? We'll feature your letters next issue.

- **Tell us your story**

Have you had a recent success (or failure) you think the wider community would benefit from hearing? Let us know.

- **Write for the magazine**

Have you got an interesting article idea? We'd love to hear about it!

GET IN TOUCH

Want to talk? You can reach us at:
contact@helloworld.cc

FIND US ONLINE

www.helloworld.cc



@HelloWorld_Edu



fb.com/HelloWorldEduMag

**SUBSCRIBE
IN PRINT
TODAY!**

PAGES 30-31



■ Preschool children will be introduced to engineering and physics concepts through creative workshops

ITALY PUTS MAKERSPACES IN EVERY SCHOOL

Ambitious plan aims to put digital computing and maker culture at the heart of every Italian school



■ Traditional laboratories in Italy are to be transformed into FabLabs

The Italian Ministry of Education has launched a national plan to put creative spaces in every school.

The National Plan for Digital Education (Piano Nazionale Scuola Digitale — PNSD) hopes to infuse a digital creative and maker culture in schools.

The ambitious plan aims to build creative workshops and makerspaces in every school. It also seeks to turn school laboratories into FabLabs (fabrication laboratories).

"Technologies have an enabling role," reads the PNSD's 'How To Design A Creative Workshop' school kit (helloworld.cc/2jmQlgJ). "They act as a digital carpet on which fantasy and reality meet."



“Serious play and storytelling will find their natural home in these areas, with a view to building cross-learning.” Workshops are built around robotics and educational electronics to promote logic and computational thinking.

Plan for workshops

Schools will learn “how to design the creative workshop, an innovative and modular space to develop the crossroads of craftsmanship, craft, creativity, and technology,” the school kit says.

school settings; not just classrooms and laboratories, but also administration, shared spaces, and informal spaces.

“It is an organic plan for innovation in Italian schools, with cohesive programmes and actions organised into five main areas: tools, skills, content, staff training, and supportive measures.”

It’s “not enough to have a laboratory and equipment to be makers,” reads the ‘How to Write The Manifesto On Use Of The School Lab’ school kit. “In addition to the safety precautions, you need to prepare and to

PNSD SCHOOL KITS

PNSD has placed all of its school kits online. These digital resources offer guidance to schools on how to create workshops and turn old laboratories into FabLabs.

The documents are in Italian but can be understood using Google Translate and other tools. They offer effective guidance for any learning environment.

THIRTEEN SCHOOL KITS ARE CURRENTLY AVAILABLE, INCLUDING:

- How to set up a laboratory
- How to design an alternative space for teaching
- How to create a shared repository of teaching materials
- How to organise cooperative learning activities: the jigsaw
- How to design a creative workshop
- How to prepare a class for cooperative learning
- How to design enlarged classrooms
- How to write a manifesto for use of the school lab

More information: helloworld.cc/2jmXjry

“ EDUCATION IN THE DIGITAL AGE MUST BE VIEWED AS A CULTURAL INITIATIVE

“Education in the digital age must be viewed as a cultural initiative,” explains the plan. “It begins with a new concept of school as an open space for learning, more than just a physical place: a springboard that enables students to develop skills for life.”

The plan is broad in scope. It will affect training and learning across all

assume certain cultural attitudes”.

The document outlines ways to prepare students for a maker workspace. These include changing the ambient air, arranging the right light, and exploring the environment. “Take your work seriously without taking yourself too seriously,” explains the school kit.

“The plan and its thirty-five actions are

also a request for collective commitment,” says PNSD, “not only from those who already work daily to create a more modern and innovative school responsive to students’ needs, but also from the communities and private stakeholders touched by the challenges that each school faces every day. (HW)



■ Heathrow Coding Challenge at the Ellen Wilkinson School for Girls, held on 8 March 2016. Image courtesy of Heathrow

THE ROYAL SOCIETY COMPUTING EDUCATION PROJECT

Respected research organisation commissioning report into computing in UK schools

The Royal Society is undertaking a research project to support schools that are teaching the new computing curriculum.

“Computing underpins almost all areas of the modern world,” explains the Royal Society. “Many new opportunities in science and engineering could not have been realised without it.”

The Computing Education Project (helloworld.cc/2jn52px) is a programme of work to gather and share evidence of the teaching of school and college computing curricula.

A new computing curriculum was introduced into English schools in September 2014. Called simply “computing”, the curriculum replaces the older ICT (“information and communications technology”) approach.

“It places a greater emphasis on

coding, algorithmic thinking, and computer science, rather than digital literacy, typing, or elementary software skills,” they explain.

“Employers are increasingly looking for students who understand computing beyond the levels of ITC,” notes the Royal Society.

However, teachers need support to implement the challenging curriculum.

Computing calls for the teaching of concepts such as algorithms and logic, plus the ability to create and debug programs from Key Stage 1.

“Teachers need support from the science community and industry to increase the number of students (especially girls) leaving school confident in coding, algorithmic thinking and computer science,” the Royal Society adds.

Inside schools

The Royal Society is working with Pye Tait Consulting to undertake research into how computing education is currently taught and resourced.

“It’s vital that we understand what’s really happening in schools, particularly in terms of teachers’ current confidence, knowledge, and skills in delivering computing education,” says a spokesperson for Pye Tait Consulting. “This will ensure that the case for future support is based on a full and honest picture.”

A report will be published by the Royal Society in early 2017.

The next stage will see detailed work programmes to deliver on the project’s objectives. The Royal Society is set to produce teaching resources, professional development opportunities, and classroom projects. (HW)

APPS FOR GOOD AWARDS ANNOUNCED



Students who won the Apps for Good Awards are now getting ready to release their apps to the public.

Six teams of students from across the UK designed apps to change the world for good. Apps For Good teams up students with professional app developers.

Fear Nothing, designed by a team of 9- to 10-year-olds from Westfields Junior School, Hampshire will help children deal with their phobias.

Another app, Changes, will help children who are too nervous to ask adults about puberty.

Debbie Forster, co-CEO of Apps for Good, explains: "We're incredibly excited to have such talented and creative students taking part, and we believe Britain's future as a tech hub is bright."

■ Apps for Good partners students with professionals who bring their ideas to life



Now in its fifth year, the Apps for Good course is being delivered in over 800 schools. Over 25,000 students were involved in 2016.

The Apps For Good website has more information on the winners, and how students can get involved in 2017: www.appsforgood.org. (HW)

MAKER ED SUPPORTS NATION OF MAKERS

The Maker Education Initiative ("Maker Ed") has announced support for the Nation of Makers organisation (nationofmakers.us).

Started by President Obama in 2014, Nation of Makers is a non-profit dedicated to helping makers through advocacy, the sharing

of resources, and the building of community.

Maker Ed already reaches 170,000 young people and family members through its Maker Corps program. It recently partnered with the Boy Scouts of America and Cognizant to bring its Young Makers framework to even more people.

It's hoped that the two initiatives will work together to put maker equipment into the hands of young people across the US.

"We strive to make it possible for every educator in America, with a particular focus on those in high-needs communities, to incorporate maker education into their learning environments in an easily accessible, highly adaptable way," says Warren (Trey) C. Lathe III, Executive Director of Maker Ed.

"We are excited by the efforts of the Nation of Makers to continue to build upon the grassroots efforts of the maker community and President Obama's Nation of Makers initiative," says Lathe.

"Maker Ed is pleased to support the Nation of Makers in its efforts to work towards a thriving, connected, and inclusive maker-centred community of practice that will further the ability of students, educators, adults, and families to engage in making and sharing." (HW)



■ MakerEd Tech Camp. Credit: Scott McLeod, Flickr



ADA ACCEPTING NEW STUDENTS

National College for Digital Skills seeking its second intake

Ada, the National College for Digital Skills, is accepting applications for its sixth-form courses starting in September 2017.

Named after the famous 19th-century mathematician and writer, Ada Lovelace, the new college is devoted to teaching advanced technical skills.

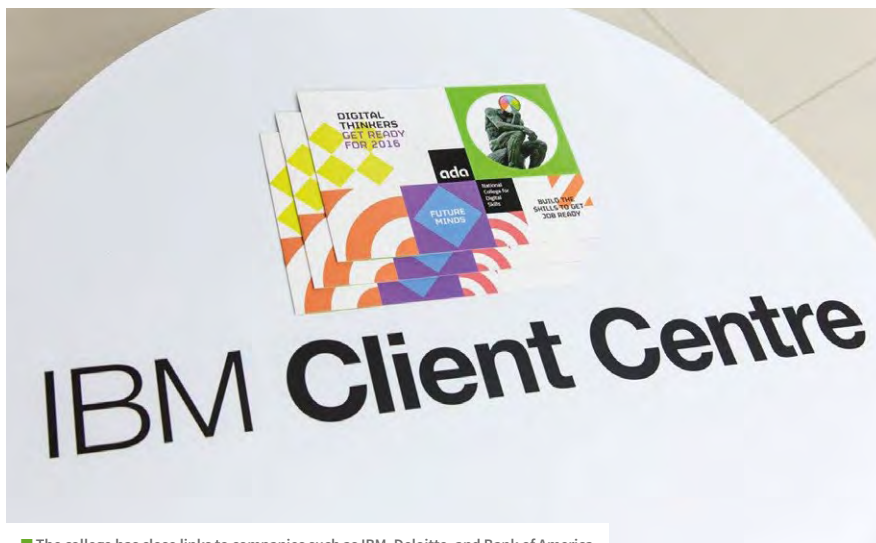
The college has close links to UK industry and has backing from several major technology firms. Those partners include Bank of America Merrill Lynch, Deloitte Digital, Gamesys, IBM, and King Digital.

"This will pave the way for a network of specialist colleges around the country," says Mark Smith, CEO and co-founder of Ada.

Based in Tottenham Hale, London, the college is well-placed to assist students from diverse backgrounds. "This is a career-catapulting education that, as a state-funded College, is free to attend," reads the Ada website.

"IBM has chosen to work with Ada because we were very encouraged by the professional approach," says Paul Milner, Senior Manager of Graduate Schemes and Learning at IBM.

Rosie Moffat, Senior Technical Manager at Bank of America Merrill Lynch, was



■ The college has close links to companies such as IBM, Deloitte, and Bank of America



ada

National
College for
Digital
Skills

BUILDING TEAMS AT ADA

"We're looking for digital explorers who can work in collaboration with others for the future technologies that society will need. And in return, we will support them in their careers and help them to flourish," says Tom Fogden, Dean and co-founder of Ada.

Students at Ada are seeded into one of three areas:

Creative: The designer whose ideas build beautiful and compelling creative products and services.

Technical: The rigorous programmer, whose logic and technical skills turn ideas into reality.

Entrepreneurial: The resilient go-getter that takes the product to the market and nurtures business relationships.

"All digital businesses need a combination of three skill sets to succeed," says Ada. "All of our students will become proficient in each of the skill sets, but will eventually specialise."

impressed by the approach to diversity. "Because Bank of America Merrill Lynch is a global company, we're very keen to employ students from a diverse range of backgrounds," she says.

"We're looking for hard-working students for our unique institution," says Smith. "Students and apprentices will have constant exposure to our industry partners throughout the work they do in the college," he tells us.

"I would encourage students to be confident that when they come through Ada, businesses like Deloitte will be really welcoming," says Marcus Williamson, Technology Partner at Deloitte.

"The digital revolution and the World Wide Web, invented by Britain's own Sir Tim Berners-Lee, are transforming the UK economy," says Baroness Martha Lane Fox, Patron. "However, the benefits are not distributed evenly, and digital businesses don't reflect the diversity of their users. This is especially disappointing because women such as Ada Lovelace and the female codebreakers at Bletchley Park in the Second World War have been so important in the development and creation of internet and computing technologies."

The course also provides enough UCAS points to move on to a top UK university or enrol on a higher-level apprenticeship.

"It's central to our ambition to turn the area into London's fastest and most dynamic centre for digital innovation," said Councillor Joe Goldberg, Haringey Council's Cabinet

Member for Economic Development, Social Inclusion, and Sustainability.

In March 2016, the Mayor of London announced £18 million of funding for the college from the London Enterprise Panel's Further Education Capital Investment Fund. (HWF)





■ A series of competitions will inspire young makers to turn their ideas into reality

PIONEERS TEENS CHALLENGE

New programme aims to keep teens interested in computing and making

Pioneers is a new programme for coding clubs and teen makers from the Raspberry Pi Foundation.

The new programme aims to keep teenagers interested in coding and digital making. It features a series of challenges



■ Pioneers will challenge teenagers to create amazing things with technology

to inspire young digital makers. Every few months, Raspberry Pi will set a new mission for the Pioneers community.

"We want to find and support teenage digital makers in the UK," says Rob Buckland, Director of Programmes. "The aim of Pioneers is to provide guidance, inspiration, and mentorship to teenage makers, and the adults who mentor them."

Young people aged between twelve and 15 will work together in teams, designing and building their ideas to solve the series of challenges. To volunteer as a mentor, visit raspberrypi.org/pioneers to register your interest.

Take the challenge

Every school term, the Raspberry Pi Foundation will set a new mission for the Pioneers community. Each challenge will have a different theme.

"There is no right or wrong way to start a Pioneers team," explains Olympia Brown, the Senior Programme Manager who will be running the programme. "It can be student-motivated or inspired by a mentor."

There is one condition. "We just ask that each team finds someone over the age of 18 to act as a mentor," says Olympia.

Pioneers will start their first mission in January 2017. "Each team has to produce a video of their work to show the judges and the rest of the world," adds Olympia.

Their projects will be judged, and prizes will be allocated to the winners. "We all like to be winners," says Rob, "but it's a great chance to get together with like-minded, creative souls and start a new community where we share skills, make our ideas a reality, occasionally blow things up, and lead the way for the future in an increasingly digital world." helloworld.cc/2jqNljG (HW)

MICRO:BITS FOR ASCENSION ISLAND

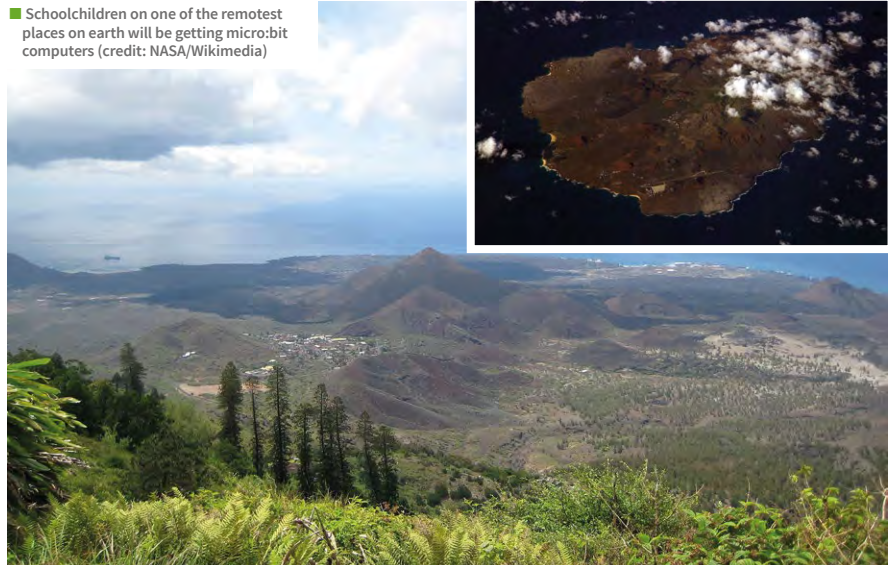
The Micro:bit Educational Foundation is sending 60 micro:bits to a school on Ascension Island, one of the remotest places on earth.

The volcanic island is a British territory in the equatorial waters of the South Atlantic Ocean. At around 1,600 kilometres from the coast of Africa and 2,240 kilometres from Brazil, it's as far away from the rest of the world as you can get.

Despite its remoteness, Two Boats Village School has around 90 children. They are all children of BBC staff employed at the BBC shortwave relay station on the island.

Alison Emerson, head teacher of the school, says: "Imagine their absolute delight at the prospect of receiving their very own BBC micro:bit as a gift to celebrate the school's 50th anniversary! No longer is our isolation a disadvantage. The children of Two Boats will be able to learn to code and make programs, enhancing their learning tenfold. It brings our children in line with 21st-century technology

■ Schoolchildren on one of the remotest places on earth will be getting micro:bit computers (credit: NASA/Wikimedia)



and the endless possibilities open to them."

Alison adds: "There are no buses or trains on the island, no big-name fast food outlets, and none of the three shops on the island

cater to the teen fashion market. By far the biggest difference is the very limited, very expensive, and very slow internet access available to young people." (HW)

PARADIGM CHALLENGE OFFERS \$150,000 IN PRIZES

Students from the United States and around the world have been awarded a total of \$150,000 in prizes in the Paradigm Challenge.

"The problem for last year's competition was how to reduce injuries and fatalities from home fires," says Michelle Lewis, Collaboration Officer. "The 100 finalists used

a wide variety of technologies to create their winning entries."

The Fire Mitt was invented by grand prize winners Emma Spencer and Scott Johnson of Bothell, Washington. It is an oven mitt that unfolds into a fire blanket. "It's such an obvious idea," says Spencer. "We couldn't believe no one had thought of it yet."

Spencer and Scott won \$50,000, along with an all-expenses-paid patent application for the Fire Mitt.

Alexis Lewis, from North Carolina, invented an "Emergency Mask Pod" system for delivering smoke masks to people trapped in the upper storeys of burning buildings.

She designed the EMP using Autodesk's Tinkercad software and then printed it with a MakerBot 3D printer. "Tinkercad is free and easy," says Lewis, adding, "3D printers are not only fun, but they're also simple to use."

"This year's challenge is to come up with new ideas for reducing waste in homes, schools, and communities around the world," says Lewis.

The deadline for entries is 1 May 2017, and there is no cost to enter. More information and sign-up information is on the Paradigm Challenge website: projectparadigm.org. (HW)



■ The Emergency Mask Pod can deliver smoke masks to people trapped in the upper storeys of burning buildings



CARRIE ANNE PHILBIN DIRECTOR OF EDUCATION AT THE RASPBERRY PI FOUNDATION

WHAT DO YOU WANT YOUR BUTTON TO DO?

Getting physical with computing can be a catalyst for creativity: the ethos of tinkering and invention is being used in the classroom to inspire a whole new generation of makers

When I first introduced the idea of making music with code to my students at Robert Clack School back in 2013 with Dr. Sam Aaron, our goal was simple: to teach computer science concepts like sequencing, repetition, conditionals, variables, and so on using simple text-based code that output sounds or music. I was so focused on that specific learning outcome that the music creation part didn't really register with me as being important. The first few lessons went well, and the students seemed excited and engaged by text-based programming; by the third lesson, we decided to introduce variables as a concept. It was a total disaster. Students were writing 12 or 13 variables to store data (in this case they were different MIDI note numbers), but then not really using them for their compositions. We had introduced an abstract concept without a real-world or immediate use case. Students were using variables for the sake of using variables, and not to solve a meaningful problem at hand. The situation could have been improved by setting a task that better motivated the introduction of variable usage, but this would have the disadvantage of moving towards a much more prescriptive approach. That day, Sam and I learned a valuable lesson which has stuck with me and guided my work ever since: give students a small handful of tools and let creativity guide the rest of their learning. Composing their own music was guiding the learning, and when students needed a construct to take their tune to the next level, we introduced it.

The all-important question

Physical computing also provides a great opportunity for creative expression: the button press! By explaining how a button works, how to build one with a breadboard attached to computer, and how to program the button to work when it's pressed, students have all the conceptual skills they need to build a thing that does something. But what do they want their button to do? Have you ever asked your students? I promise it will be one of the most mind-blowing experiences you'll have if you do. Amy will want her button to take a photo, Charlie will want his button to play a sound, Tumi will want her button to explode TNT in Minecraft, and Jack will want his button to fire confetti out of a cannon! (Doesn't he always?) Idea generation is the inherent gift that every child has in abundance. As educators, we're always looking to see how engaged our students are in the subject matter we're teaching, and young people are never more engaged than when they have an idea and want to implement it.

Allowing this kind of free-form creativity and tinkering in the classroom obviously has its challenges for teachers, especially those confined to rigid lesson structures, timings, and small classrooms. The most common worry I hear from teachers and parents is "what if they ask a question I can't answer?". Encouraging this sort of creative thinking makes this almost an inevitability. How can you facilitate roughly 30 different projects simultaneously? The answer is by using



those other computational and transferable thinking skills: problem solving, iteration, collaboration, and evaluation. Clearly specifying a problem, surveying the tools available to solve it (including online references and external advice), and then applying them to solve the problem is a hugely important skill and this is a great opportunity to teach it.

Hands-off guidance

When we train teachers at Picademy, we group attendees around themes that have come out of the idea generation session. Together they collaborate on an achievable shared goal. One will often sketch something on a whiteboard, decomposing the problem into smaller parts; together the group will divide up the task to work on. Each will look online or in books for tutorials to help them with their step. I've seen this behaviour in student groups too, and it's very easy to facilitate. You don't need to be the resident expert on every project that students want to work on. The key is knowing where to guide students to find the answers they need. Curating online videos, blogs, tutorials, and articles in advance gives you the freedom and confidence to concentrate on what matters: the learning. Outside of formal education, events such as Raspberry Jams, CoderDojos, CAS Hubs, and Hackathons are an ideal venue for seeking and receiving support or advice.

Cross-curricular participation

The rise of the global maker movement, I think, is in response to abstract concepts and disciplines. Children are taught lots of concepts in isolation that aren't always relevant to their lives or immediate environment. Digital making provides a unique and exciting means of bridging different subject areas, allowing for cross-curricular participation. I'm not suggesting that educators should throw away all their schemes of work and leave the full direction of the



Children are taught lots of concepts in isolation that aren't always relevant to their lives or immediate environment

computing curriculum to students. However, there's huge value in exposing students to the possibilities for creativity in our subject. Creative freedom and expression guide learning, better preparing young people for the workplace of tomorrow. Occasionally we need to stop and ask, "What do you want your button to do?" **(HWW)**

Carrie Anne Philbin is Director of Education at The Raspberry Pi Foundation, a Python Software Foundation and Computing At School board member, author, and YouTuber.

#INSIGHTS



LEARNING THROUGH MAKING

Learning through making. It's part of the fundamental philosophy of 'Constructionism' behind Papert's ideas, and key to a practical subject like computing. On the surface it seems very simple; in a subject based on making things, students learn by doing just that. Yet dig deeper, and the idea of learning through making has some much wider implications to explore.

Making learning fun

It's pretty clear if you know young people that making is something that's going to engage them. Active lessons always get the popular vote from classes, especially if they let students make choices about what they work on. The sense of achievement you get from making something and sharing it with others or taking it home is pretty motivating too. There's always a few who would rather have a 'theory lesson', but the engagement you get from making is usually a powerful motivator. It's hugely important to get people engaged with learning for it to be successful, but learning is more complicated

than simply paying attention to something. Seeing learning through making as only a way to engage people would be missing something much deeper than that. For proponents of Constructionism, it's also about how making interacts with the way we develop understanding.

From concrete to abstract

Our culture of education in the West can often be very focused on the cognitive; the abstract thinking that can be clearly defined in learning objectives, exams, and books. We tend to think of formal education as the process of coming to understand abstract





ideas, with abstract ideas being the most important level of understanding that can then be applied to our everyday lives. Young children usually start learning about numbers through physically playing with concrete objects such as blocks, counters, and toys, but the aim is for them to move on to being able to discuss and manipulate numbers as abstract ideas. Dealing with concepts

well in the work of Jean Piaget, almost universally taught in teacher education courses across the Western world.

Affective learning

Whilst we see the cognitive side of learning as key to understanding, we tend to see the affective, or experiential and feelings-based, as something useful for making learning engaging

of gears allowed him to develop an affective understanding of how machines work, and realise that these complex constructions are knowable and understandable. Mark Surman, CEO of Mozilla, describes this memorably as seeing the 'Lego lines' in the world; the visible joins that help you understand that something was made by a person, and that with the right learning that person could be you.

“ PAPERT WRITES ABOUT CHANGING HIS WORLDVIEW, NOT ONLY IN TERMS OF GAINING KNOWLEDGE, BUT IN GAINING A NEW RELATIONSHIP WITH KNOWLEDGE

totally on an abstract level is hard, and often children have to return to these concrete methods to support their understanding. It takes time before children can add and subtract without the convenient aid of fingers to count on; even when this is mastered, they often return to counters when learning about the more complex concept of division. This trajectory from understanding concepts in concrete, real life terms towards being able to explore them in the abstract is explored

and memorable, but not a fundamental part of it. Papert saw this differently. In 'Mindstorms' he vividly relates the affective experience of playing with cogs and gears as a child, and how he came to an understanding that machines could be both very structured but also creative ways of interacting with the world.

Papert writes about changing his worldview, not only in terms of gaining knowledge, but in gaining a new relationship with knowledge. Manipulating and exploring the concrete objects

Learning as becoming

Such a change in understanding is a bit of a shift from the way educators are often encouraged to see learning; it's a different metaphor for the process. Much of the time our language about learning is based on what Prof. Anna Sfard calls the 'learning as acquisition' metaphor, where learning is seen as discrete blocks of content that can be gradually acquired. Paulo Freire pejoratively called this the 'banking model'. There are other metaphors; when exploring the potential of learning through making it helps to think about the 'learning as becoming' metaphor, the idea that we learn in order to explore and develop who we are as a person, and the way we see our identity fitting in to the world. ▶

► New tools for learning

Much of this could be an argument for learning through experience, but for Papert it was using computers that he described as being incredibly powerful. Why? Computers allow us to manipulate abstract concepts in a way it simply isn't possible to do in the physical world. Logo may seem like primitive software to us in 2016, but Papert saw its potential to allow children to actively manipulate concepts such as angles and geometry. This made abstract concepts accessible to children to manipulate and understand by feel, much as a sand and water tray in the early years allows children to explore their understanding of basic physics. We expect children to move on from this playful, exploratory approach to learning as they get older, but perhaps this is only because we lack the tools to make more sophisticated concepts concrete and accessible to them to manipulate. The power of computers for learning is described in Papert's writing not as being a way to deliver content to children, but as a tool they can use to explore and manipulate previously abstract concepts in a concrete way.



Harnessing the tools

Making is often a fun and engaging way to learn, yet its power can go beyond engagement and towards a very different way of learning and understanding the world. It takes a shift in how we think about learning and in the way we encourage young people

to use computers to understand the world. These days, we certainly have more powerful and sophisticated tools accessible to young learners; perhaps the biggest challenge is understanding how they can be used not only to engage, but to learn in new ways that are both effective and affective. (HW)

WORKING MEMORY

In education we draw from so many fields of understanding, including our own subjects, the field of education studied in our teacher training, our own experiences of learning, and working to support others to learn. One field that has had quite some impact on thinking about education in the last few years is cognitive psychology, as its studies of how the mind works have much to give us in thinking about how students learn.

One such key insight is 'cognitive load theory'. This states that we have a limited working memory of around seven bits of information. This information is fragile, only lasting a few moments, and so it's imperative for learning that we can transfer information to long-term memory which is much more robust. Experts in a subject have lots of facts, ideas, and schema about an area in long-term memory. When they see

a problem, they can relate the information about it to what's in their long-term memory and solve this problem, using working memory only for the very specific details of the problem, rather than the structure of it or the background knowledge needed to understand it.

is new to the learner, and some features of a programming language they haven't yet met, and you can be providing a challenge that far exceeds students' working memory limitations.

One solution to this with lots of backing in research is the use of worked examples.

“ STUDENTS WHO LEARN A NEW CONCEPT AND THEN ARE GIVEN SEVERAL WORKED EXAMPLES TO READ HAVE BEEN SHOWN TO DO MUCH BETTER

Computing and programming are subjects in which there's often a lot of new information. Combine a problem to solve that has a structure never seen before with contextual information that

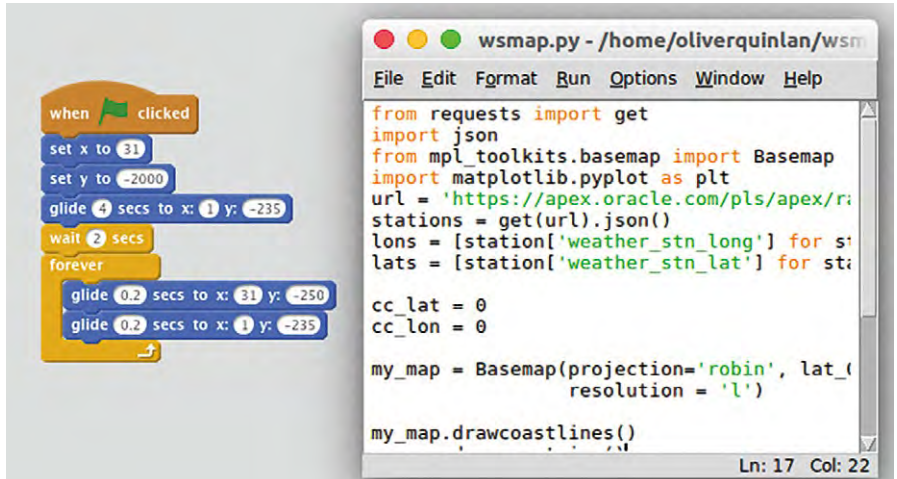
Students who learn a new concept and then are given several worked examples to read and study have been shown to do much better when presented with similar problems to solve afterwards. Worked examples can

BLOCKS TO TEXT:

COMPUTATIONAL THINKING TO PEDAGOGICAL THINKING

A regular area for debate amongst teachers of programming is how to best support the transition from visual programming to text-based languages. Visual languages and environments such as Scratch are a hugely powerful tool for introducing students to the concepts of programming, allowing them to explore concepts, solve problems, and create products through programming. However, it's important to get the experience of working in text-based languages, both to cover the curriculum and to develop the skills for the next stage.

Dorling and White explore this transition in their paper 'Scratch: A Way to Logo and Python', and their research has suggested that we might want to think about this 'transition' in a different way. They explored approaches including unplugged, visual, and textual programming, and the ways students engaged with problem-solving in these contexts. They discovered that it was valuable to think about these different media as pedagogical tools, with different strengths for teaching and learning, rather than stages students had to progress to and leave the others behind.



The power and variety of text-based programming gives prevalence to this form, but in terms of teaching it's the various concepts and skills that we're trying to lead students through in the best way possible. This research shows that problem-solving can start in the realm of 'unplugged approaches' where the nature of problems can be explored, students move into visual languages as a form of pseudocode, and then to text-based programming to develop

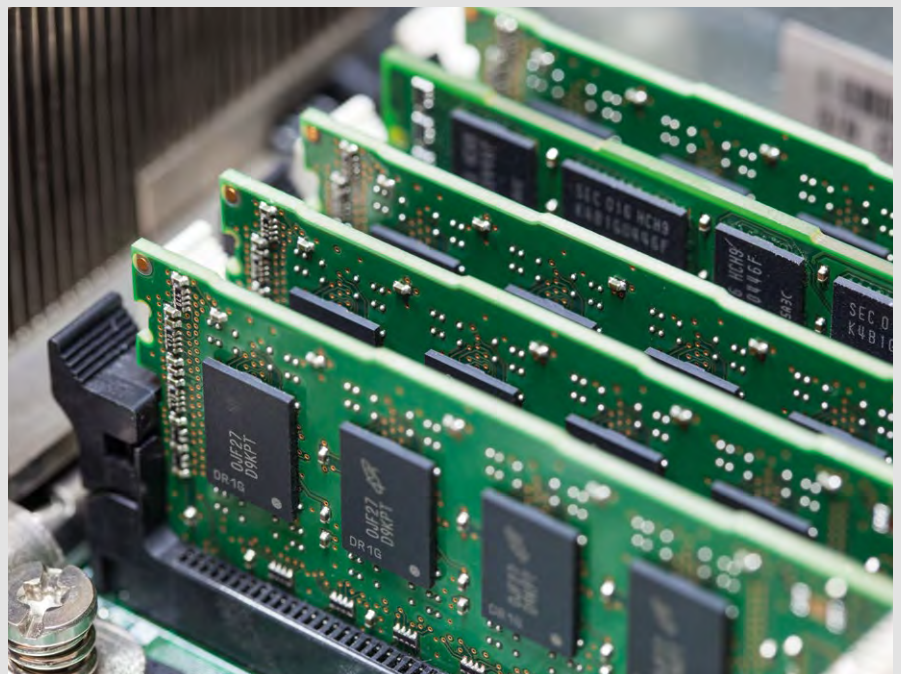
a deployable solution. The transition here is more about the pedagogy and the learning than moving from one type of tool to another. These studies provide a way of looking at this issue that will be new to many people, and they suggest a different way of approaching it with your students.

For more, see Dorling and White's paper 'Scratch: A way to Logo and Python' at helloworld.cc/2iHfsyx. (HW)

look like we're taking the problem-solving out for the students, but it's important that they have the opportunity to understand problem structures and commit them to long-term memory if they are to be really successful at solving problems as a result of your teaching.

Think carefully about how much new information your students have to hold in working memory at any one time, and hone in on the individual concepts that you really want them to learn and focus your teaching on, transferring those to long-term memory. Traditional, worked examples are one proven way to help this to happen.

For more, see Chapter 16 of Hattie and Yates's 'Visible Learning and the Science of How We Learn' or dig deeper into Sweller & Cooper's paper on worked examples in Algebra at helloworld.cc/2iH2kti (content is behind a pay wall). (HW)





LORNA ELKES DEPUTY HEADTEACHER

THE WORLD IN A BOX

How using VR headsets to support learning helped teachers to become more confident with the range of the computing curriculum

How often have we wanted to go travelling? We know the benefits of a real-life experience to engage learning; it's why visits and trips are so valued. But there's financial limitations to organising them. Google Expeditions lets you be temporary explorers. We tried out the Google Expeditions Pioneer Program (helloworld.cc/2jrjBy).

Despite having little time to prepare or understand what it involved, top marks go to the teaching staff who went with the technological flow to see where it would take them.

Ease of use

We went from Rome to London, from the oceans to Mount Rushmore, and to space. What makes the experience manageable is that it's controlled by one person. You become the tour guide: children can freely explore or have items of interest pointed out. The Control Tablet pauses the journey, allowing the children to return to the real world. The staff received a brief training session, yet managed the equipment comfortably. All children could take part, even revisiting areas to ensure nothing was missed; all adults noted the impact of this. The follow-up work was of a high standard, particularly for those whose life experiences are rarely beyond their neighbourhood.

As expected, this generation of 'digital natives' were at ease with the equipment, even if most hadn't seen or used a VR headset before.

It's not just about the coding

We're now a 'Google School' using only cloud storage, with raised expectations of computing. Adults can now reflect on the breadth of the English computing curriculum; it's not just

THE HEADSET

The cardboard Google VR headset was sturdy, and able to accommodate a range of devices with simple fastenings. No straps meant no fussing. It also meant they were quick to dive into or share. The Google Expeditions can also be followed via a tablet for children unable to access the headsets.



about coding. Not bad, considering many primary/secondary colleagues are 'digital immigrants'. Even new teachers don't always feel confident delivering many aspects of computing.

Keeping it simple is important

Our journey towards a functioning network with working devices is recent, yet we stay true to our 'unplugged' roots. Younger pupils begin by making algorithms with each other, first as human robots, then with Bee-Bots. Next, they debug other groups' programming while using Read Write Inc for technical vocabulary. There's many excellent resources from Barefoot (barefootcas.org.uk) and Phil Bagge (code-it.co.uk), to be used or adapted for unplugged teaching. Maintaining this approach has worked well, helping pupils' understanding of programming principles rather than being language-specific. It's also supported teachers' own development. **(HWR)**

Lorna Elkes leads technology for learning and has a keen interest in how it can engage pupils across the curriculum.

MARK THORNBUR TEACHER

CODING THE CAESAR CIPHER

The Caesar Cipher is one of the simplest, and oldest, systems for cryptography. Let's see how it can be implemented very simply in Python...

One of the simplest methods to create secret messages is undoubtedly the Caesar Cipher. As you might expect, it's named after Julius Caesar, who used it in his correspondence. To encipher a message, we choose a whole number and shift every letter down the alphabet by that number of places. For example, if we choose 6 then A becomes G, M becomes S, and Y becomes E; after Z we start at A again. To decipher the message, we just shift the letters back again.

Modular arithmetic and ASCII codes

To code this in Python, we'll use remainders. If we number the letters from 0 to 25 (always count from 0 in Computer Science!), then we just add the shift factor. If we then take the remainder after dividing by 26, 26 becomes 0, 27 becomes 1 and so on, meaning that A comes after Z.

In Python we can turn a letter into a number using

the `ord()` function, which produces the ASCII code corresponding to a letter. Unfortunately, it doesn't give a number between 0 and 25. Upper-case letters are between 65 and 90, while lower case letters are between 97 and 122. The other numbers are for punctuation, control characters, and so on. To deal with this, we can employ a nice trick called conjugation. We move our numbers back to start at zero, perform the shift, then move forward again. Finally, we convert back to a letter using `chr()`.

See the box for a short Python program to accomplish this.

Working with files

Those who know a little more Python might like to adapt this to take input from a text file and output to another text file. This lets you encipher a whole letter or even a book in a single pass. I like to set this challenge at Christmas and give my students a copy of Dickens' "A Christmas Carol" from Project Gutenberg (gutenberg.org). This website has text files of many classic out of copyright books, in several languages.

Cracking the code

This method of creating secret messages is not very secure. Short messages can be deciphered by just applying all 25 possible shifts and reading the output; longer ones can be attacked by a method known as frequency analysis. We look for the most common letter in the message and assume this must correspond to the most common letter in the English language, e. This gives us the shift and we can now read the message.

Next issue, we'll look at how to perform frequency analysis on a file. [\(HW\)](#)

Python code to implement a Caesar Cipher

```
plainText = input("Enter your text:\n")
shift = int(input("Enter how many places
to shift:\n"))
cipherText = ""
for char in plainText:
    pos = ord(char)
    if 48<= pos<= 57:
        newpos = (pos-48+shift)%10+48
    elif 65<=pos<= 90:
        newpos = (pos-65+shift)%26+65
    elif 97<=pos<=122:
        newpos = (pos-97+shift)%26+97
    else:
        newpos = pos
    cipherText += chr(newpos)
print("Coded Message:")
print(cipherText)
```

Mark Thornber has been a maths teacher at Durham Johnston for the last 25 years. Mark has been interested in the mathematical parts of computing since owning his first ZX81.

PAPERT'S

LEGACY

Co-creator of Logo, pioneer of programming in schools, and godfather of the maker movement. Seymour Papert has had an immense impact on digital making, mathematics, and CS education

Seymour Papert died in July 2016, leaving behind a legacy of profound impact on so many aspects of education. He provided much of a generation with their first experience of computer programming through the Logo language, particularly its pioneering implementation of turtle graphics. He also developed many of the ideas that lie at the foundation of computing education and digital making. He was the first to coin the term 'computational thinking'; he recognised that there was little point to teaching children to program as an end in itself, but that through their learning to program they would start looking at problems, other subjects, and the world quite differently. He moved beyond Piaget's view of learning as through experience to the theory of 'Constructionism'. He foresaw the impact that providing children with

access to the world's knowledge would have for the nature of schooling; and finally, he was an advocate for equitable access to cheap digital technology for all.

So much of what we're learning about good practice in computing education was figured out by Seymour Papert 30 or 40 years ago. Papert was one of the first to recognise that a young person could make things in their mind through making things in the world.

Here, four contemporary educators look back on Papert's work, and draw out some of the lessons we can learn from this today. We begin with Dr. Gary Stager, veteran teacher, educator, speaker, and colleague of Papert for twenty years. Gary curates the Papert archive at dailypapert.com and is co-author of the highly recommended *Invent To Learn – Making, Tinkering, and Engineering in the Classroom*.



PAPERT

FATHER OF THE MAKER MOVEMENT

Written by: Gary S Stager, Ph.D.



Papert was not only a recognised mathematician, artificial intelligence pioneer, and computer scientist; he was also the father of educational computing and the maker movement.

By the late 1960s, Papert was advocating for every child to have their own computer. At a time when few people had ever seen a computer, Papert believed that children should program them. They should be in charge of the system; learning while programming and debugging. He posed a fundamental question still relevant today: "Does the child program the computer, or does the computer program the child?" Along with colleagues, Papert created Logo, the first programming language designed specifically for children and learning. Logo dialects, like Scratch and Snap!, are still in use fifty years later.

Papert's legacy extends beyond children programming. In 1968, Alan Kay was so impressed by children's work in Logo he sketched the Dynabook, the prototype for the modern personal computer, on his flight home. LEGO's line of robotics gear is named after Papert's seminal book, *Mindstorms*. In 1993, Papert conjured up images of a knowledge machine that children could use to answer their questions.

Making things and making meaning

As students expressed formal mathematical ideas of how they wanted the robotic Logo turtle to move about in space, it would drag a pen (or lift it up) and move about in space as a surrogate for the child's body; they were learning not only powerful ideas from computer science, but constructing mathematical knowledge by "teaching" the turtle. From the beginning, Papert's vision included physical computing and using the computer to make things that lived on the screen and in the real world. This vision is clear in a paper Cynthia Solomon and Seymour Papert co-authored in 1970-71, "Twenty Things to Do with a Computer." (see box). This made the case for the maker movement more than forty-five years ago.

Computing for all

Social justice and equity was a current running through all of Papert's activities. If children were to engage with powerful ideas and construct

20 THINGS TO DO WITH A COMPUTER

"In our image of a school computation laboratory, an important role is played by numerous "controller ports" which allow any student to plug any device into the computer... The laboratory will have a supply of motors, solenoids, relays, sense devices of various kinds, etc. Using them, the students will be able to invent and build an endless variety of cybernetic systems." (Papert and Solomon, 1971)



knowledge, then they would require agency over the learning process and ownership of the technology used to construct knowledge.

"...Only inertia and prejudice, not economics or lack of good educational ideas, stand in the way of providing every child in the world with the kinds of experience of which we have tried to give you some glimpses" (Papert and Solomon, 1971)

One laptop per child

It frustrated Papert that kids couldn't build their own computers. In 1995, Papert caused a commotion in a US Congressional hearing on the future of education, when an infuriated venture capitalist scolded him while saying that it was irresponsible to assert that computers could cost \$100, have a lifespan of a decade, and be maintained by children themselves (<http://helloworld.cc/2jr6do7>). Later, Papert would

be fond of demonstrating how any child anywhere in the world could repair the \$100 OLPC laptop with a single screwdriver. The Raspberry Pi finally offers children a low-cost programmable computer that they may build, maintain, expand, and use to control cyberspace and the world around them. (HWW)



The One Laptop Per Child XO-1- one of the many projects inspired by Papert. ◀



THE PATRON SAINT OF MAKING AND CODING

To support children's learning, we must give them opportunities to design, create, experiment, and explore. Logo, Scratch, and the maker movement do just that.

Written by: **Mitchel Resnick, Professor of Learning Research, MIT Media Lab**

Seymour Papert has served as inspiration for every educational technology project I've worked on: LEGO Mindstorms, Computer Clubhouses, Scratch, and more. Seymour was a true visionary, recognising possibilities and opportunities decades before others. The projects and ideas that he developed, starting in the 1960s, laid the intellectual foundation for today's maker movement and Learn to Code movement. I think it's fair to consider Seymour as the patron saint of making and coding.

Looking beyond technology

To understand Seymour's contribution, it's important to look beyond the technology. Seymour is probably best known for his Logo programming language, the first programming language for children. But what's more important are the ideas underlying Logo. Seymour's Constructionist theory of learning provided a new vision of how children learn, and how we can support their learning. Seymour argued that children learn best when they are actively engaged in making things and expressing themselves.

Seymour's Constructionist theory has guided our work on Scratch. On the MIT Scratch Team, we sometimes talk about our approach in terms of the "Four P's of Creative Learning," all of which build on Seymour's ideas.



A Scratch project honouring Seymour Papert, created by Scratch community member eduardm. See the full project at helloworld.cc/2jrdCUu ▼



Four P's of Creative Learning

Projects. Seymour worried that schools too often introduce students to a disconnected set of concepts and skills. Instead, Seymour advocated a project-based approach to learning, in which students learn concepts and skills in the context of meaningful projects. Rather than introducing coding through a series of puzzles, Scratch supports children in turning their ideas into games, stories, and other projects.

Passion. People often think that children want things to be easy. Seymour knew otherwise. He recognised that children are willing to work hard, and tackle difficult problems, when they were working on projects connected to their interests. He called this "hard fun." In Scratch, we support many different types of projects, since we know that children have many different interests. We view the incredible diversity of projects on the Scratch website as a sign of success.

Peers. Seymour was inspired by the samba schools in Brazil, where people come together to create music and dance routines for the annual carnival festival. We see Scratch as a type of online samba school. We created the Scratch online community at the same time as creating the programming language, since we recognised the importance of children learning with and from one another.

Play. Seymour embraced a playful approach to learning, encouraging learners to try new things, experiment, take risks, and learn from failures. We designed Scratch to support playful tinkering. It's easy for children to snap together programming blocks, take them apart, and playfully experiment with new possibilities.

PAPERT AND SCRATCH

A few years ago, I gave a conference presentation about the Scratch programming language, discussing what children learn as they create and share projects in Scratch.

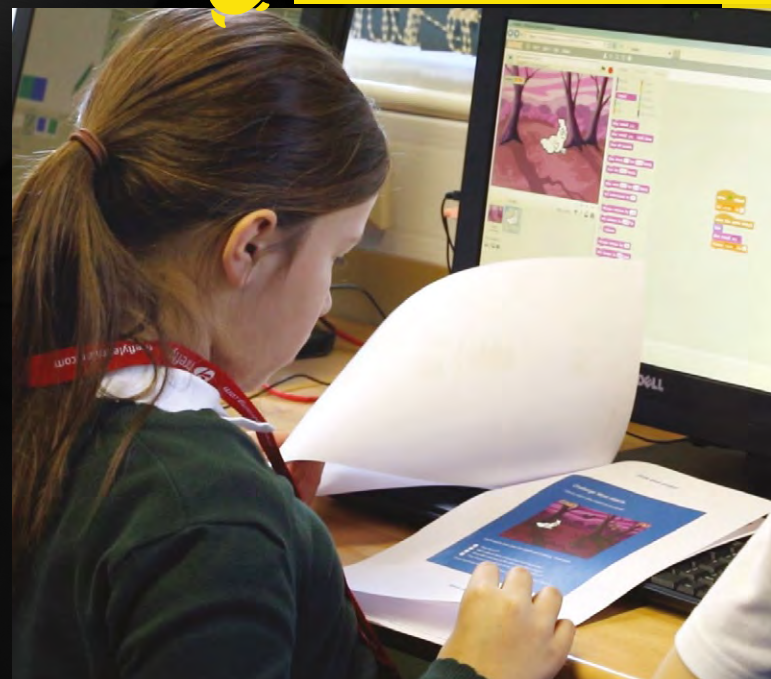
After my presentation, in the Q&A session, someone stood up and asked: "Wasn't Seymour Papert trying to do these same things 20 years ago?" The comment was meant as a critique: I took it as a compliment. I answered simply: "Yes."



“ Seymour is probably best known for his Logo programming language, the first programming language for children

Putting ideas into practice

It's not always easy to put Seymour's ideas into practice, but it's worth the effort. I will be happy and proud to spend the rest of my life trying to turn Seymour's visions into reality, and I hope others will too. (HW)



COMPUTATIONAL THINKING AND LOGO: ANOTHER PERSPECTIVE



Papert's Logo was great at the time, but it's not without problems: microworlds are limiting, it doesn't support types well, and it can encourage tinkering rather than thinking

Written by: **Greg Michaelson**

The Logo language embodies Seymour Papert's Constructivist notions of a microworld as a locus for explorative learning. Thus, at its simplest, Logo is based on sequences of commands to control a turtle, which leaves a trace as it moves around a plane.

Papert was a visionary of inclusive computer use: beginners find Logo's turtle microworld engaging and motivating. We can see Logo's spirit in contemporary graphical languages which enhance what is essentially the turtle microworld with avatars, colour, and sound.

The limits of microworlds

Alas, I think that the wider Logo language, like many graphical languages, is a poor fit for contemporary ideas of computational thinking. Manipulating a given microworld through coding - that is, assembling programs from pre-made commands - is certainly an excellent starting point for beginners. But, quite quickly, we want them to start building their own microworlds through problem-solving and programming. These are part of manipulating a pre-made microworld, but the problems are bound by the entities and operations that the microworld offers. Once we try to go beyond turtles, Logo is really quite impoverished.

Papert thought children should acquire the capacity to "think like a computer"; that is, to follow step-by-step procedures, and to construct programs in a microworld by themselves, first acting out appropriate sequences of commands. Hence, the core Logo design is strongly procedural, with poor support for thinking about information structures

Logo and Lisp

Furthermore, Logo is very much a creature of its time and place. Much of Logo feels as if Lisp, its MIT stablemate, has been bolted onto the turtle world. Lisp is a perfectly decent language once you've got your head round its eccentricities, but I think it's a poor choice as a teaching language. In particular, Lisp has an unsatisfactory notion of type, which

LOGO AND TYPES

```
to ffind :v :l
  if empty? :l [output "fail]
  if :v = first :l [output 1]
  output 1+ffind :v (butfirst :l)
end
```

Logo, like Lisp, is weakly typed, so beginners don't start with the concept of how operations and values may be combined. Weak typing places a strong onus on run-time testing. For example, the code here returns a number or a string, depending on whether the item is in the list or not.

Logo inherits; this makes it hard to progress from microworld assemblages to crafting one's own new microworlds.

Lisp's basic entities are atoms, conflating numbers, strings, and identifiers. These are directly reflected in Logo's words, which must subsequently be disambiguated by different sorts of quoting and use contexts. And, what's worse, Logo inherits Lisp's one-size-fits-all lists, with a confusing vocabulary of operations.

Bricolage and hacking

Papert was keen on a "bricolage" style of problem-solving, driven by exploratory changes to a poor solution to try to find a better one, so Logo's weak types may have been a virtue for his pedagogy. But, as programs grow, this approach can lead to misguided hacking, which can prove frustrating and demotivating for impatient beginners. Rather, we would like learners to make hypotheses about how their programs should behave, and reason about why they don't.

Latterly, Papert himself recognised this tension. In the preface to the 2nd edition of *Mindstorms*, he expressed concern that he might have unintentionally encouraged a classroom focus on "structured programming", at the expense of his wider pedagogy of "thinking about thinking". (HW)

TODAY'S CHILDREN'S MACHINE

For many children and young people, the use of technologies is integral to their experience and understanding of family and social life, and to learning.



Written by: **Josie Fraser**

Within developed countries, technologies are now a part of the everyday life of most people. Making a clear distinction between real world and virtual activities is increasingly irrelevant. While different environments offer different affordances, digital and physical spaces typically coexist.

The sum of all human knowledge?

When asked about Wikipedia, Jimmy Wales famously said, "Imagine a world in which every single person on the planet is given free access to the sum of all human knowledge. That's what we are doing." As one of the most visited sites in the world, Wikipedia, along with Google, YouTube, and Facebook, provide internet users with unprecedented access to information, and have changed how we learn.

In an age where young people can access, if not the sum of all human knowledge, then more information than has been available to people at any point in history, the urgent question becomes "how can we equip them to make the most of it?" How are we responding to the mainstreaming of web and mobile technologies within education?

Digital literacy

Digital literacy is not just about technical ability. The definition I most frequently use is digital literacy = functional technical skills + critical thinking + social

Children's machines today. How should we develop young people's criticality alongside their skills as users of technology? ▼



THE ROLE OF SCHOOL

Children and young people are still not routinely supported in developing the critical competencies they need to navigate their world. Some countries have begun to support learner digital literacy at national level, and some have started to recognise the need to enable school staff to develop the skills and confidence necessary to support learners.

engagement. It's about critically and creatively engaging with online content and communities. Understanding bias, evaluating information, and checking facts become critical in a world where children and young people are not dependent on schools, parents, or carers for accessing or verifying knowledge. We need to ensure young people are developing the ability to navigate critically, review the content they come into contact with, find the content they need, and make use of what's available to them.

Creativity, identity, community

The increasing opportunities for young people to engage creatively with computing and coding are very welcome, along with the availability of affordable devices, connectivity, and low-cost computers like the Raspberry Pi. There's still more to do in relation to how we understand and champion young people's creativity online. There's no doubt that their activity online - creating, collaborating, and sharing - are creative acts. Importantly, young people are developing their identities online, and finding their voice and their place in the world. They create and recreate themselves: their tastes, views, values, and passions. By recognising the wide range of creative and social practices young people take part in every day online, and understanding the role of technical skills and awareness, we can perhaps start to see how we can help their habits positively contribute to both their own development, and to their physical and digital communities. (HW)

This launch edition of Hello World includes more content inspired by Papert's work: Oliver Quinlan gives his own views on why Logo apparently failed to change the nature of education, and Prof Michael Kölling, Phil Bagge, and John Stout share ways in which Greenfoot, Scratch, and Snap! have built on Logo's heritage.

(HW)

SUBSCRIBE

Sign up today for a year - prices start at FREE!



**FREE
IN PRINT**
For UK-based
educators!

Subscribe Today

- Get all 3 of 2017's term-time issues
- Have it delivered directly to your door
- Hello World is not available in stores!

(Hello World)

HOW TO SUBSCRIBE

- Visit us online:
helloworld.cc/sub1year
- Call our subscriptions hotline:
+44(0)1202 586848



Not a UK-based educator?

- Buy any issue for £6

Visit: **helloworld.cc/buyissue**

- Subscribe from £15 for 3 issues

Visit: **helloworld.cc/sub1year**

SUBSCRIPTION FORM

YES, I'd like to subscribe to **Hello World** magazine!

This subscription is: ☐ For me ☐ A gift for someone

HW#1

YOUR DETAILS ☐ Mr ☐ Mrs ☐ Miss ☐ Ms

First name Surname

Address

Postcode Email

Daytime phone Mobile

Are you a UK-based educator? ☐ Yes ☐ No

GIFT RECIPIENT'S DETAILS ONLY ☐ Mr ☐ Mrs ☐ Miss ☐ Ms

First name Surname

Address

Postcode Email

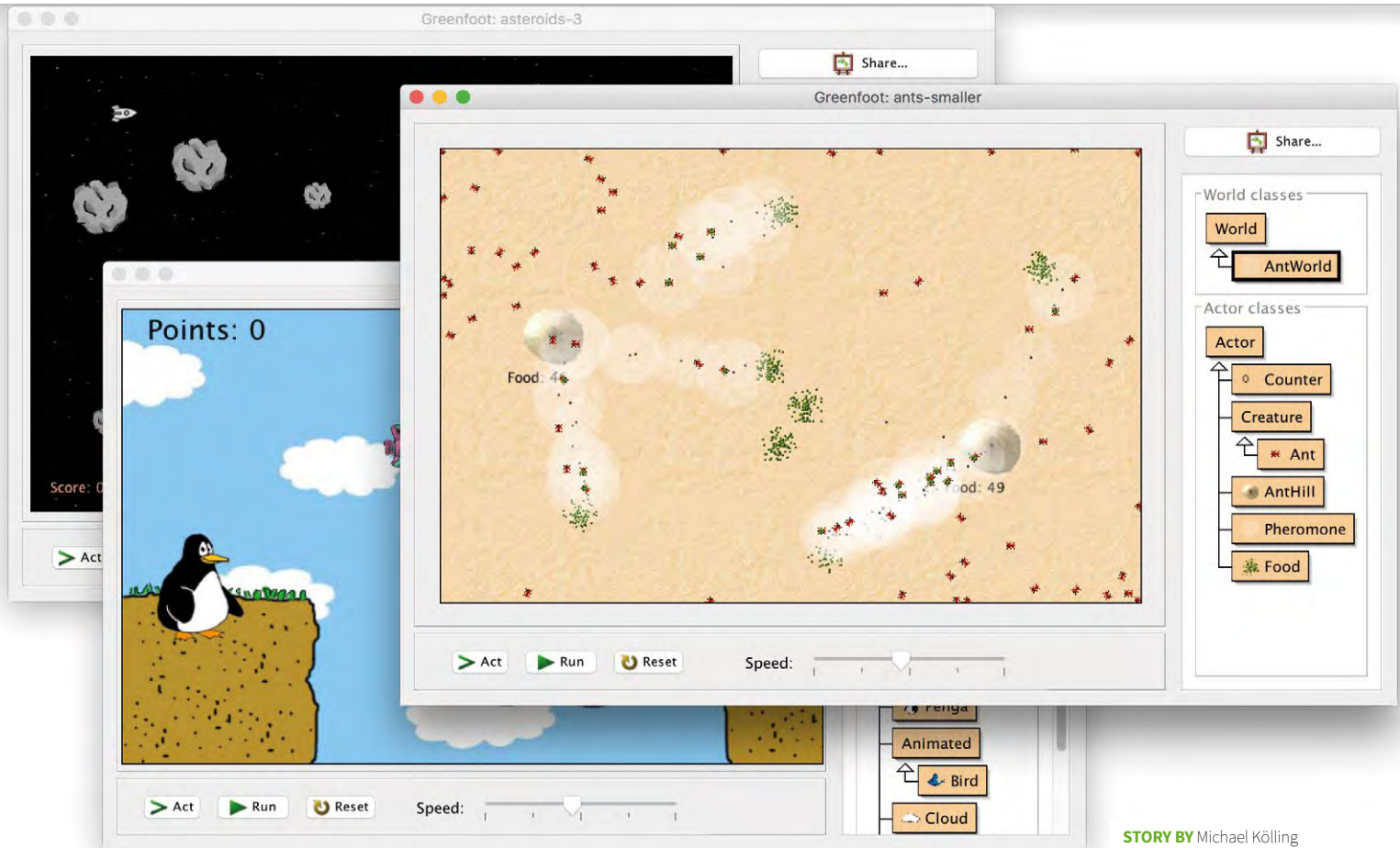
Is the recipient a UK-based educator? ☐ Yes ☐ No

If you're giving Hello World as a gift, please complete both your own details (left) and the recipient's (right).

Your Hello World subscription will start from the first issue, unless you specify the issue number here:

RETURN THIS FORM TO: Hello World Subscriptions, Select Publisher Services Ltd, PO Box 6337, Bournemouth BH1 9EH. We'll be in touch!

☐ Please tick this box if you DO NOT want to subscribe to the Raspberry Pi Foundation Education newsletter (no spam).



STORY BY Michael Kölling

GREENFOOT AND WHAT PAPERT'S TURTLES TAUGHT US

Modern programming environment, Greenfoot, is strongly influenced by Papert's constructionism. Michael Kölling discusses some underlying principles, and how you can use them in your teaching...

Seymour Papert, one of the early and most influential pioneers in computing education, died a few months ago on 31 July 2016. His work continues to influence computer science teaching, including the design of our own system, Greenfoot. Papert's influence on computer science education is profound and varied, most notably centring around constructionism, his theory of learning. In this article, however, I want to concentrate on one specific concrete contribution: his development of turtle graphics.

Many of you, if you're my age, will fondly remember Logo's turtle graphics, one of the earliest and most successful software systems developed specifically for the learning of programming. And even if you're too young to have lived through the popular time of the turtles, you're likely to have heard the name mentioned.

Turtle graphics became so well-known because it embodied principles that are fundamental and timeless, that changed the way in which beginners could learn the underlying concepts of programming.

And even though the original Logo implementations of turtle graphics are dated now, these principles remain, and are still highly relevant.

For us as computing teachers today, it's interesting to identify what these principles are, and how we can still make use of them in modern programming systems.

A short history of turtles

The history of turtle graphics is very closely tied to the Logo programming language, developed in 1967 by Seymour

Papert and others. Papert added turtle graphics to Logo in the late 1960s, and although Logo was a general-purpose programming language, the turtles are perhaps its most enduring legacy. Although we know the turtles today as a software simulation in a graphical micro-world, the first turtles were in fact real robots, rather chunky in today's terms (Figure ?); they moved along the floor with a pen attached, leaving lines behind wherever they went. By programming the movement of these robots, users could produce drawings on the floor.

The physical "turtles" were soon accompanied by simulated ones that did a similar thing on a graphical display: drawing lines in reaction to movement instructions. Using a combination of move, turn, and repeat commands, surprisingly intricate pictures could be created.

After Papert described his ideas in his classic book *Mindstorms: Children, Computers, and Powerful Ideas*, turtle graphics was re-implemented for a number of other programming languages; today, we can find implementations for almost every language we choose to use.

The big ideas

So what was so different about turtle graphics that made it so successful?

The basic idea of turtle graphics was that the programmer programs a turtle to move on the screen (or paper), using move and turn commands, leaving visible traces behind. By combining these commands with generic programming concepts, such as conditionals, repetition, parameterisation,

execution of a program into a kinaesthetic exercise. These are powerful ideas, and it's hard to understand today how revolutionary this was at the time.

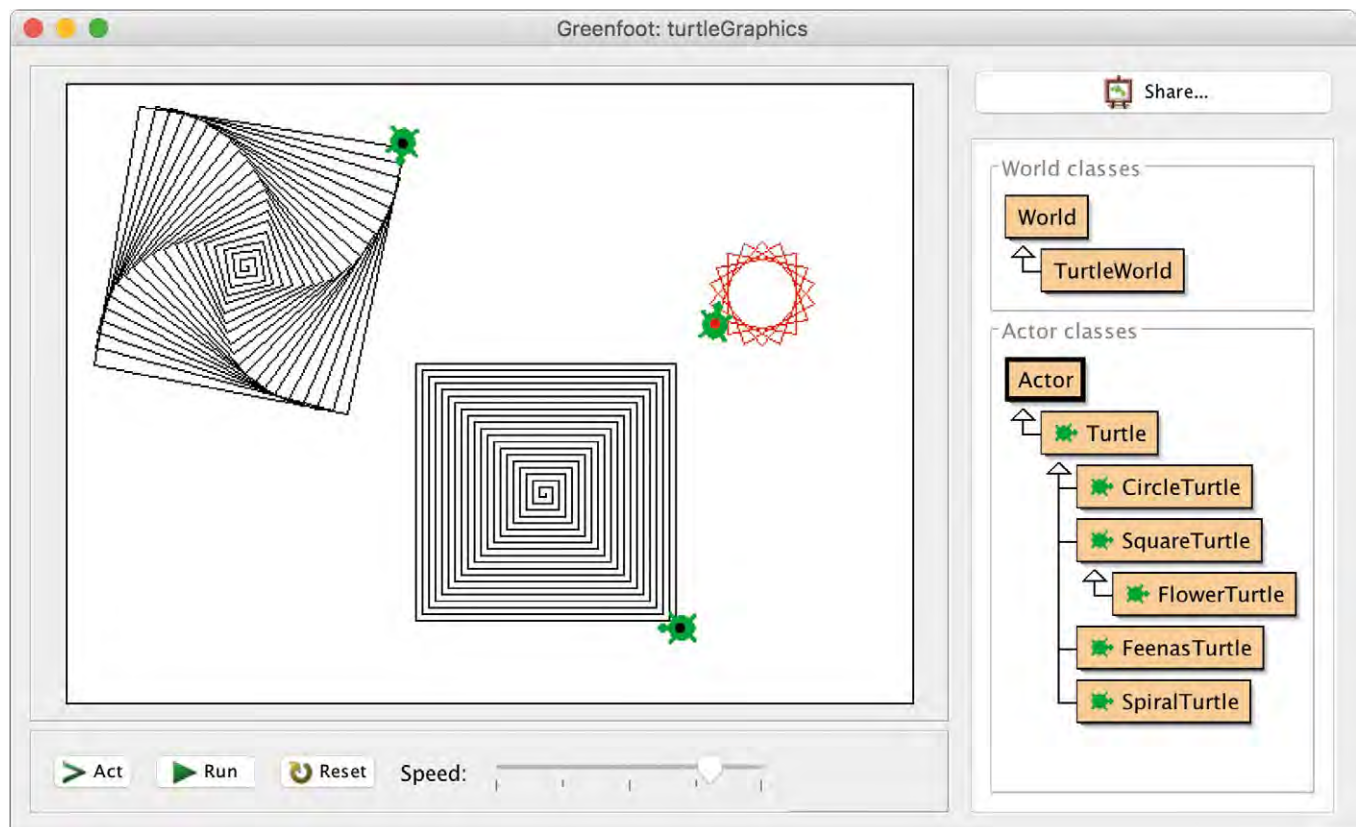
Turtle graphics was a pioneer in taking programming away from pure mathematics and numbers, turning it into a visual and visible activity. Reasoning about a program, until then highly abstract and formal, became concrete. Perhaps

“ TAKING PROGRAMMING AWAY FROM PURE MATHEMATICS AND NUMBERS, TURNING IT INTO A VISUAL AND VISIBLE ACTIVITY

and subroutines, programming fundamentals and general problem-solving could be learnt. Papert later coined the term "body-syntonic reasoning" for this kind of thinking: learners could imagine themselves in the position of the turtle, playing through their program as a first-person simulation. They could debug their program by stepping forward and turning just as they imagined the turtle to do, turning the

most importantly, users could suddenly see a program execute. While most programs at the time took some input and then produced some output, the process between these two remained hidden and inaccessible. With turtle graphics, you could suddenly watch your program producing its output. ▶

■ A turtle graphics scenario, implemented in Greenfoot.





■ Seymour Papert with one of his turtles: robots that moved and created drawings with an attached pen

- Debugging became partly implicit. Instead of requiring organised testing, it was often the surprised reaction of “Why did it do that now?” that pointed to a bug in the program.

This visual quality was combined with interesting tasks (the images it produced were often intriguing) and a limited

Papert's ideas in modern systems

With the advent of modern graphics and video games, our turtles have perhaps lost some of their engagement and fascination qualities for the younger generation. Papert's ideas, however, are as relevant as ever. Luckily, there are a number of modern educational programming systems available that build directly on these principles, and package the same qualities into environments that provide more modern functionality.

The best-known of these are perhaps Scratch and its later variant Snap! These environments provide micro-worlds that not only exhibit the same qualities discussed above – visualisation, engagement, and extendibility – but also provide support for experimentation and discoverability that hugely surpass original turtle graphics systems.

In this article, however, I will discuss one system in particular that owes much of its design to the same principles: Greenfoot (www.greenfoot.org), a modern micro-world framework.

Greenfoot

Greenfoot is an educational programming environment designed to achieve the same goals that Papert formulated 45 years ago. Programs developed in it are highly visual, usually constructed

goes far beyond traditional turtle libraries. Some of the most relevant aspects are as follows:

- Greenfoot provides an integrated environment, including the editor, compiler, and runtime system. This greatly aids in program development and debugging. It de-emphasises the tools (editor, compiler, etc.) and lets users concentrate on the task.
- Greenfoot is fully object-oriented. This not only teaches a more modern programming style than older turtle graphics implementations, but also allows for the creation and simultaneous execution of multiple turtles (or other actors), allowing development of much more interesting examples.
- Greenfoot provides interaction facilities that allow direct interaction with individual objects, providing more fine-grained experimentation than older systems.
- The editor in the Greenfoot system provides modern support for program development, such as inline help, code completion, and edit-time error checking, thus combining Papert's ideas with modern advances in tool development.

But most importantly, Greenfoot is not a micro-world, as turtle graphics is, but technically a micro-world meta-framework. This means that it's a system in which countless micro-worlds can easily be created. Turtle graphics-like systems are just one example, but just as easily we can write traffic simulations, video games, predator-prey simulations, board games, card games, chat clients, or any number of other examples (Figure 3). Greenfoot takes Papert's lessons and generalises them to a whole class of applications: anything we can think of that produces two-dimensional graphical output as its result.

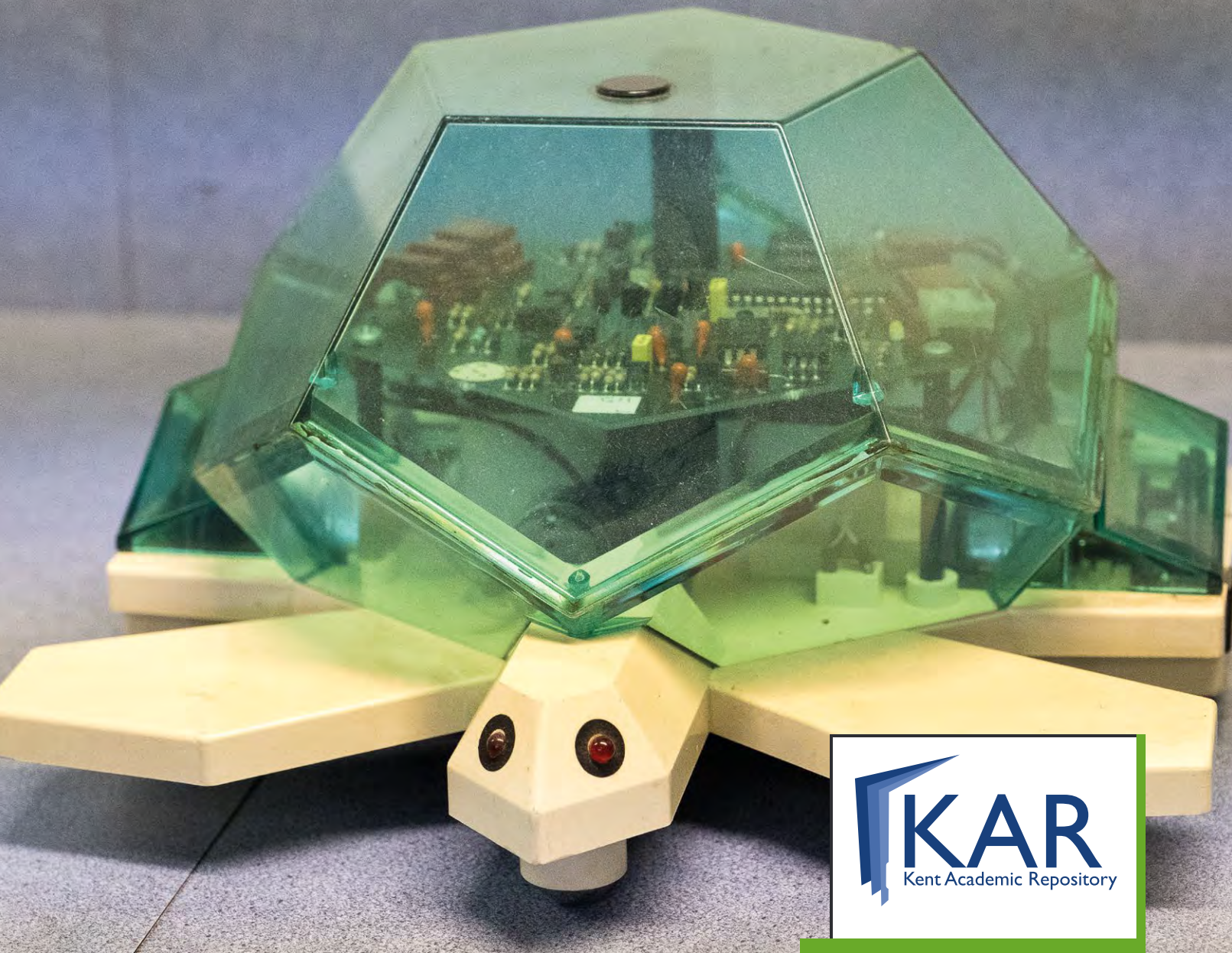
“ THIS VISUAL QUALITY WAS COMBINED WITH INTERESTING TASKS (THE IMAGES IT PRODUCED WERE OFTEN INTRIGUING) ”

problem domain. Both of these aspects help to keep learners engaged and feel safe and in control.

A last aspect that I personally always found very important was that it was carefully designed to teach composition as well as use of commands. By providing, for example, a turnLeft command but no turnRight, users were prompted to build such a command themselves (by defining a procedure). Thus, it was made clear from very early on that, as programmers, we are not only users of language, but also creators of language.

around visible movement of actors on screen, allowing engaging examples and visualisation of program execution. It uses a general-purpose programming language (Java or Stride), provides a well-defined problem domain, and is extendable.

Turtle graphics, for example, can easily be programmed in Greenfoot (Figure 2). A teacher could provide a functional turtle, and learners could go through the same activities that Papert's pupils went through. However, Greenfoot also provides additional functionality that



FURTHER READING

A full discussion of Stride is outside the scope of this article; I will leave that for another day. However, if you are curious, you can find information online <http://blogs.kent.ac.uk/mik/stride/>.

A more detailed history of the development of Greenfoot and its sister system, BlueJ, is available in the article *Lessons From The Design of Three Educational Programming Environments: Blue, BlueJ, And Greenfoot*, available from <https://kar.kent.ac.uk/56662/>

graphics, and the pedagogical benefits transfer directly. This powerfully illustrates the fundamental nature of Papert's ideas and the lasting influence he continues to have on our teaching. Using Greenfoot or similar systems, you can combine modern technology with powerful pedagogy. 

Teaching with Greenfoot

If you're not familiar with Greenfoot, you can easily find examples and material online. Once you start to look around, you will quickly see the wide variety of examples that can be created.

Good starting points are a series of video lessons titled "The Joy of Code" (blogs.kent.ac.uk/mik/joy-of-code-table-of-contents/) and the Greenroom (greenroom.greenfoot.org), a community of teachers interested in Greenfoot. In the Greenroom, you can find lesson plans, schemes of work, projects, slides, and much more. But most importantly, you can talk to other teachers and the designers of Greenfoot to get help, discuss ideas, and share your own material. A large amount of material, as well as the software itself, is freely available to teachers and learners.

Stride

One specifically interesting aspect of Greenfoot is Stride. While Greenfoot can be programmed in standard Java, programs can also be written in the more recently published language Stride. Stride provides a stepping stone between block-based languages such as Scratch and traditional text-based systems. As such, it may provide an ideal system to facilitate the transition between the two.

In summary

Users of Greenfoot don't usually think of the system in terms of turtle graphics; visually it seems far removed, and in functionality it has progressed a long way since Logo was designed. Educationally, though, the legacy is clear. Constructivism underpins the design of Greenfoot, just as it has shaped the design of turtle

IS COMPUTING EDUCATION IN ENGLAND BECOMING MORE EXCLUSIVE?

Who is studying computer science at English schools? The data paints a worrying picture...

STORY BY Peter Kemp and Billy Wong

2 014 saw the introduction of a new Computing curriculum in England, followed by new computer science qualifications for 16- and 18-year-olds and a deadline for the phasing out of the old ICT qualifications. We welcomed the change in the curriculum, but would the new computer science qualification confound expectations and have girls taking it in similar numbers to the old ICT? Would we have schools in poorer communities offering it? Or is computer science a socially exclusive subject?

Many teachers report stories of students from tough backgrounds finding well-paid jobs in the tech industry, based on

a recognition of their skill rather than the name of their school or the wealth of their parents. On the face of it, computing is a meritocratic profession; no one should care about who is writing the code as long as it's well written. It shouldn't really matter what sex you are, or the colour of your skin, or whether you have a disability. Sadly, the reality of the tech industry is the dominance of white middle-class men (and some studies even suggest that it helps to have a beard). One question that needs to be answered is whether this disparity in the industry is being established much earlier, in pre-university computing education.

more closely into the data, especially who these students are. Our findings raise some concerns.

GCSE and ICT are different

The Roehampton report shows that the distribution of students who took ICT at GCSE is roughly representative of the general population, where 27% of students receive pupil premium; this was not the case for computer science, with only 19% of pupil premium students. In terms of state school provision, grammar schools were far more likely to offer computing than non-selective providers (53.1% vs. 31.7%). We also observed a marked disparity between regions when looking at students taking GCSE computing: 6.5% of students in the South East compared to only 4.2% in the North East. Urban schools were also more likely to offer computing at GCSE or A level than those in rural locations (29.5% vs. 22.7% and 25.1% vs. 18.1% respectively).

Striking results were also obtained when looking at the ethnicity of the students taking computer science. Asian and Chinese students were over-represented in the population of students taking GCSE computing; in contrast, black students were substantially under-represented. This mirrors the patterns observed in the American education system and the reality of the tech industry itself.

CLASS ACTION IN EDUCATION

A lot of work has been done to promote female role models in technology, but where are the working-class and ethnic minority role models?

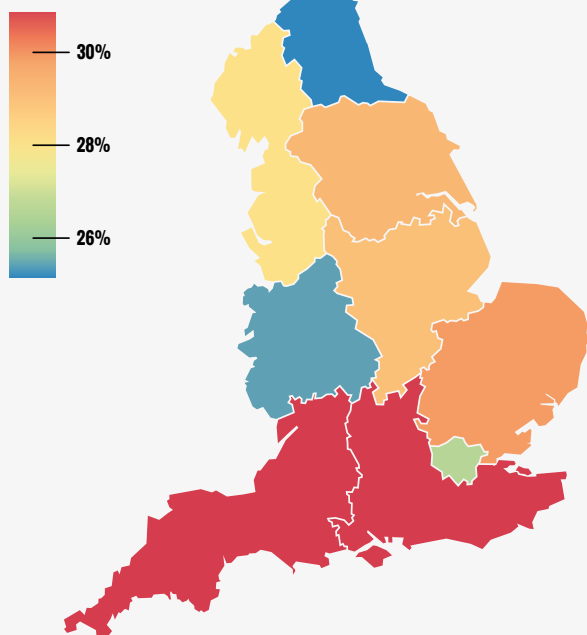
A question you might like to ask your students is "what do Ada Lovelace, Grace Hopper, Bill Gates, Mark Zuckerberg, and Alan Turing have in common?" An obvious answer might be that they all speak English and are white, but another would be that they were all privately educated.

The Roehampton annual computing education report

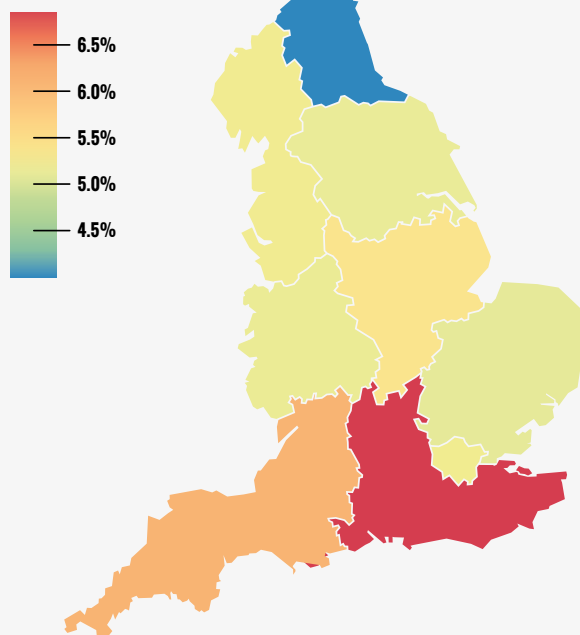
Using government data on student results for the 2015 exam year, we looked into the schools and students taking computer science and ICT qualifications. Whilst concerns about the low uptake in computing qualifications amongst female students are well-known, very little is known about other social factors. The 2015 Roehampton annual computing education report (helloworld.cc/2jmYuY4) looked into some of these factors. We found the numbers of students taking GCSE and A-level computer science have been increasing steadily over the last few years. However, we also looked

GCSE COMPUTING REGIONAL HEAT MAP:

BY REGIONAL SCHOOL



BY REGIONAL STUDENTS



Gender and computing

When looking into gender issues, we found that over 40% of students taking GCSE ICT were female, but for the new computer science qualification

combine with gender and social class. For example, while few black boys, as a proportion, took GCSE computer science, the percentage for black girls is higher than the average. Chinese



WE NEED TO ENSURE THAT ALL STUDENTS WHO WANT TO STUDY COMPUTING WILL BE OFFERED THE OPPORTUNITY

this number was below 20%. Looking into the gender characteristics of a school, maybe unsurprisingly, all-girl schools were less likely to provide computer science than all-boy and mixed schools. More worryingly, in mixed schools, a quarter of GCSE and nearly two thirds of A-level computer science providers had no females in their groups. Interestingly, amongst the girls who take GCSE computing, a higher percentage of them are on pupil premium than boys in the same circumstances. The picture is also mixed for ethnicity, when we

students, regardless of social class or gender, are very well represented in GCSE and A-level computing in terms of proportional representation.

Our data suggests that computing is still a socially exclusive discipline, with fewer girls and fewer students on pupil premium forming part of the cohort, and with some ethnic minority backgrounds heavily under-represented. We need to ensure that all students who want to study computing will be offered the opportunity to do so, and that they will be supported in their aspirations.

PARTICIPATION

How does computer science participation compare with traditional science subjects? It appears the issues around gender, socioeconomic background, and race/ethnicity are also prominent in the physical sciences, especially physics. Most young people still imagine the scientist in very stereotypical ways - a very clever person in a lab coat, typically a man, maybe with glasses (or even 'crazy' hair), and perhaps socially awkward. These images are not all that different to the standard computing or geeky stereotypes.

How teachers can help:

- If you have entry requirements to study computing, look into how they impact on different sections of your student cohort.
- Try to promote role models in your classroom that include female, working-class, and ethnic minority professionals.

How schools can help:

- GCSE computing should be offered in all English secondary schools, regardless of region, gender characteristics, or admission policy. (HM)



HOW TO MAKE A VISUAL EFFECTS ARTIST

Visual effects feature in movies, TV shows, games, and animations. The people who make VFX are collaborative artists, with a blend of science and art skills.

Visual effects (VFX) in movies and TV have become commonplace, as digital tools and technologies have developed in the last 25 years. The photorealistic quality of VFX is now so convincing that the dividing line between live-action and animated movies like Disney's *Jungle Book* has become completely blurred. Is it a live-action movie because of the boy actor, or is he a performer in an animated movie? For the artists making VFX, the labels are less important than the creation of stunning visual entertainment across all genres of film and TV.

MAKING VFX LOOK REAL

Guardians of the Galaxy relies heavily on animated characters to tell its story. VFX artists at Framestore created and animated two of the five hero characters. Check out the 'Rocket Reel' at helloworld.cc/2iAM6WS to see how the animated Rocket character was made in a computer, and the filming reference of how Rocket should look in each scene.



Image courtesy of Heyday Films and Framestore

Introducing students to VFX

VFX for movies start with a visual idea based on a narrative. Teachers who are looking to introduce VFX to their media students can start by asking students to identify what they see as an effect in favourite films or scenes. How do the effects work with the story? Use show-and-tell reels on the web to help students dissect how some scenes have been made (there's a list of reels and effects on the www.NextGenskillsacademy.com website). Once the students can identify some basic techniques like stop-motion animation or green-screen, they can try their hand at making something.

Let's make a VFX!

Ask your students to pitch ideas for a VFX-dependent story or animated movie to each other. Teachers should agree with students which ideas can be produced. If students want to make a stop-motion or CGI animated movie, the genre will lead you to use certain tools. Check out stopmotioncentral.com for animation, or blender.org for CGI. For stories and films that require live-action filming, read Mark Sawicki's 'Filming the Fantastic'. It contains lots of professional information, but a little research and planning can make all the difference to the finished film! Encourage your students to work in teams. They

should identify roles that they can each fulfil. Creating VFX requires a blend of skills and job roles, not just sitting around in glorious isolation!

Progression and future careers

Making VFX is a craft, and practice makes perfect! At the end of each movie or sequence that your students make, they will have something for their portfolios. Employers are less interested in qualifications and more in seeing what applicants can do. Encourage students to identify their best work, replace old with new, and reflect on how their skills are developing. Portfolios are important for students who want to progress to further creative industry studies, whether through vocational education or academic study routes.

Today, the main employers for VFX skills are film and TV companies, but new applications for VFX skills are emerging. Virtual and augmented reality, biomedical, and architectural visualisations are just some of the potential career areas that a mastery of VFX skills can lead to. helloworld.cc

Phil Attfield VFX and Animation
Partnership Manager at NextGen
Skills Academy

PAUL POWELL TEACHER

INTERACTIVE FICTION AND THE LOVE OF CODE

Paul Powell is piloting a programming unit using Quest: a text-based adventure game engine. He argues that creating these games helps develop understanding...

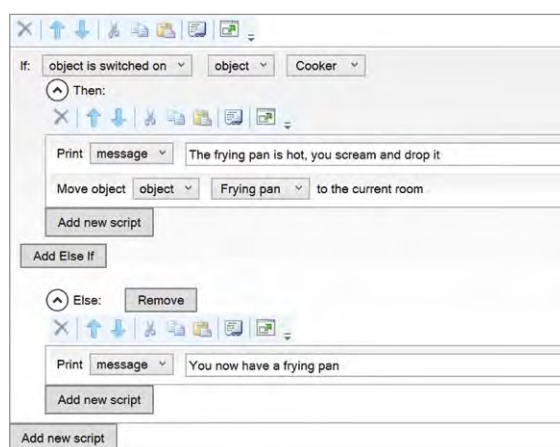
How I teach programming doesn't seem to work for many pupils. They follow the class, they learn the skills, and some even choose computer science for GCSE; they still don't love it, though. That's a blind spot for me: I do, on occasion, code for pleasure. My enthusiasm can carry the lesson, but the students won't then go home and try it.

The coding activities I have been running in class were based on altering or writing small programs with outcomes that were quite convergent: asking students to complete tasks with defined outcomes that teach a concept. Those that found coding intrinsically interesting were hooked; the others followed along. With little teaching time, we barely had the space to be creative with code.

Code fragments

Quest (helloworld.cc/2iD8q11) enables every student to create a game from their imagination. In the first lesson, they used point-and-click to create a working game with several rooms to move between and a number of objects to look at.

Having invested in the world, they want to enrich it; programming shows them how. The world already works



■ A script for picking up a frying pan: if the cooker is on, you scream and then drop the frying pan

The Wizard's Castle

My first experience of coding was on an old game called The Wizard's Castle. My brother taught me to alter the code so that the game said "you stepped in a poo" rather than a puddle. Being about 10 years old, this was fascinating power. Most of the game was written for me and I altered bits I understood, heralding a world of possibilities.

and is enhanced by tiny fragments of code. Students can use and adapt these to accomplish simple things, practising programming techniques over and over to achieve their personal goals.

Creativity and constraints

One major difference I've noticed is that the restrictions of text-based adventures mean that students aren't overreaching as much with what they want to accomplish. I suspect we've all seen the student that wants to create a complex game in Scratch and is disheartened that it seems hard! I've had questions along the lines of "how do I make it so that when they drink the milk they find a message at the bottom of the jug?" or "how do I make it so they see more when I turn the light on?", and so on. These questions lend themselves to thinking about the abstractions and code involved in the game: the rooms, objects, verbs, sequence, selection, and so on.

Enhancing the game through these little scripts means that everyone ends up with a working game, and that introducing one new feature doesn't mean significant restructuring of the rest of the code. Rather, you are dealing with smaller problems in isolation and based on already solid abstractions.

Not the answer you were looking for

This is, of course, not "the answer" to teaching coding and algorithms, but for me it seems like I've found another piece of the puzzle: writing programs that make sense to students because they thought them up! **(RPW)**

QUANTUM: TESTS WORTH TEACHING TO

How do you know if your students have understood a concept? Ask them questions.

STORY BY Simon Peyton Jones

A well-posed question makes you think. It leads to dialogue, exposes misconceptions, and helps distinguish when you think you know something from when you actually understand it.

Suppose you had a ready source of good questions about computing. You could use them to check your own understanding. You could set a quick overnight quiz, which your students could do on their phones, so that the next day you'd have an idea of whether they'd "got it". If not, their answers might tell you their misconceptions.

Writing good questions is hard

The trouble is, it's hard to come up with good questions. It's easy to think of ones that test whether you can remember Python syntax, but harder to find ones that exercise computational thinking. But there's good news: we only need do it once. A corpus of

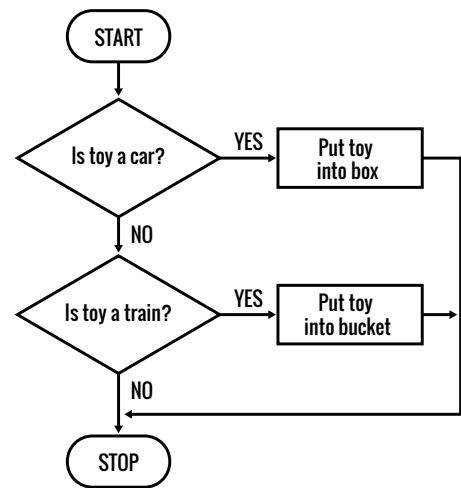
COMPUTING AT SCHOOL

EDUCATE · ENGAGE · ENCOURAGE

Kate collects toys using this algorithm. She has a toy in her hand.

Identify the selection statement that matches this flowchart.

- A** if
- B** if elseif
- C** if else
- D** if elseif else



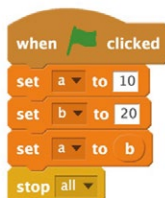
■ Testing if students really understand the difference between selection statements. What misconceptions would the wrong answers here suggest?

HOW YOU CAN HELP

We need your help, because the network effect is key. The more people using Quantum, the more people will think it's worth writing questions for it: the more questions there are, the more attractive it will be to use. So whether as an author or consumer of high-quality computing questions, we need you. Start here: helloworld.cc/2jAJXL3

Consider this Scratch program. What will be the values of a and b after it has finished?

- A** a = 30
b = 0
- B** a = 20
b = 10
- C** a = 20
b = 20
- D** a = 30
b = 20



well-crafted, free, online questions would be a useful tool for the computing curriculum, year after year.

The Quantum project has precisely this goal. We aim to develop, with your help, thousands of questions on the computing curriculum, from primary to A level. We have 2,500 already, and more come every day.

What makes Quantum different?

Quantum is different from other online platforms. It's focused on frequent, low-stakes, formative, diagnostic assessment to support learning (in contrast to high-stakes summative assessment).

It's also school-led and crowdsourced. Teachers use the questions on the system and upload their own.

Quantum uses a free online platform, Diagnostic Questions. Moreover, the questions will be available online, free,

forever; anonymised data will be available to researchers.

It's evidence-driven and research-led. Our partners include two leading assessment experts: Tim Oates (Cambridge Assessment) and Robert Coe (Durham Centre for Evaluation and Monitoring). CEM aids to provide quality control for the crowdsourced questions, by analysing the data from thousands of students doing thousands of questions. No one has ever done this before.

The project has two goals; the first is being immediately useful to computing teachers. We have a need for high-quality assessment material; Quantum will produce this quickly.

Secondly, no one has tried to crowdsource assessment items, and then use data to evaluate and improve their quality. If we can make this work, the results will be useful for all subjects in any country. We aim to change the world! **(HW)**

PROJECT QUANTUM - A TEACHER'S PERSPECTIVE

Iain Davis, Assistant Head and Year 6 teacher, has taken a look at diagnosticquestions.com and Project Quantum, and predicts a bright new future for assessment.

STORY BY Iain Davis

INTRODUCING THE CONCEPT

When I introduce something new to staff, I always approach one of our teachers who is less confident with technology. If they can manage it, the rest should too, right? After a quick 10-minute demo with a less confident KS2 teacher, she took to it like a duck to water.

You get an e-mail message saying you have won a prize in an internet raffle. You should:

- A. Be very pleased and tell your friends.
- B. Just delete it because it is not true.
- C. Contact the sender to check if it is true.
- D. Send your home address so that your prize can be delivered to you.

Something I hear again and again from my fellow teachers is 'there's a big problem with assessment in schools'. From changes in the national curriculum to high-stakes end-of-year tests, the whole system is under pressure. Computing isn't like reading and maths and so quite often I feel it can't be measured by children sitting a paper. They do enough of that in the run-up to SATs, and don't our teachers already have enough to do?

I was introduced to Project Quantum in October 2016, so our school is still at the start of our journey with it. We had been looking for an effective way to assess computing for a while, with mixed success. We wanted something to help us measure progress and provide easy assessment of learning. The online platform diagnosticquestions.com does this perfectly. It has many questions across a number of subjects; the current count is 29,284, with 2,340 being computing.

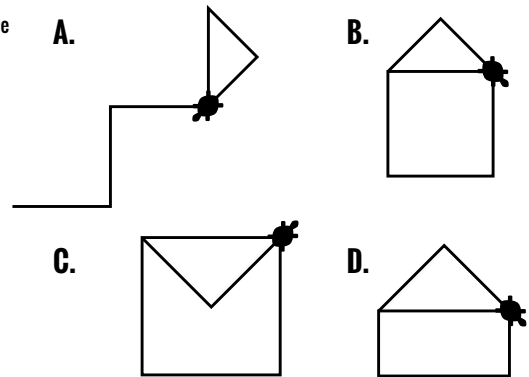
■ A great way to test (and teach) programming is by getting pupils to work through code in their heads or on paper.
helloworld.cc/2jADjo7

A turtle is facing right.



What shape will the following sequence of Python instructions draw?

```
forward (100)
right (90)
forward (100)
right (90)
forward (100)
right (90)
forward (100)
right (45)
forward (70)
right (90)
forward (70)
```



Improving teaching and learning

The biggest benefit I've found from using the site is that not only does it give me a way to measure progress, but it also shows me quickly where my students are on their learning journey. I'm then able to push those that need to be moved onto working at a greater depth, and also know who I need to support more. Normally with computing, it will take a few sessions to get a good grasp on what it is that pupils can and can't do.

Free and easy-to-use

The platform is free to sign up to and on their site they state that it will be free forever! No need to set aside precious budget areas. As is always the way with new initiatives, there are early adopters and those that are less keen to embrace change.

The children have taken great enjoyment from it, as they get instant feedback and they see it as a low stakes/low threat piece of assessment, unlike certain booklets that we give many children in the month of May. At present, I'm trying to come up with an end-of-year question bank for each year group so that we can measure their progress across the school.

In control

This platform also puts the power into the teacher's hands. I choose what I will assess, and, in turn, that reflects what I will teach and what my children need to learn. What I love about it is that it is easy-to-use, quick to set up, and the insights I get into what my children know is invaluable. Will this new platform change attitudes to assessment? I hope so. (IWD)



OLIVER QUINLAN SENIOR RESEARCH MANAGER

FAILURE TO START? THE LEGACY OF LOGO

Raspberry Pi's Oliver Quinlan discusses Logo's influence, 40 years on...

A powerful learning language ahead of its time, Logo was designed in the late 60s and spread to classrooms by the 80s. Its reach and familiarity today shows its success in its goal of fundamentally changing education systems.

When many think of Logo, it's the turtle guided by typed instructions. 'Turtle graphics' is so iconic it's become the definition of Logo. However, it's much more than that.

Logo, from the Greek for 'thought', was designed as a tool for children to manipulate ideas in the computer, to interact in a concrete way with abstract knowledge, like finding the properties of angles by drawing shapes. To draw a triangle, a child must experiment with instructions and work out that the angles must total 180 degrees. But 'turtle graphics' was just one feature. It was designed to let children manipulate language, so that abstract knowledge in many subjects could be explored with computers.

Powerful ideas

Logo was more than a way of learning programming. It was a radical educational philosophy; children learned by exploring complex concepts on their own. Seymour Papert described its revolutionary potential in books that questioned the nature of schools and teacher-led lessons. He wanted to put learning in the hands of children, empower them to discover ideas themselves, and he believed that in Logo, this tool was being developed.

Papert wanted to see Logo in every classroom; in the USA and UK this was successful. Logo descendants still exist, yet education systems are detached from its

underlying approach to learning. Why is this?

Researchers have explored how Logo was adopted over many decades. Maybe its success on one level prevented its success on another. Turtle graphics was so compelling for teaching geometry and simple programming that it was taken up just for learning these. Thus, it was hard to shift perceptions of it to a tool with much deeper implications.

Learning from the legacy

Educators developing the teaching of computing can learn from this. Turtle graphics was easily understandable and educators were ready to adopt it, yet this acceptance made it hard for many to see the true potential of the tool. As we work to bring the learning opportunities that computers bring, we should communicate this in ways that don't undermine the tools' true potential.

Failure, or just the start?

Some would say Logo failed to realise its ambitions, but success depends on perspective. Without Logo we wouldn't have Scratch, making coding accessible to young children, and governments wouldn't have adapted curricula in response. If Logo's story is a 40-year one, perhaps it failed, but I suspect this story will be much longer... 

Oliver Quinlan is Senior Research Manager at the Raspberry Pi Foundation. He works to better understand how people learn and make with technology.

BCS SCHOLARSHIP SCHEME

BCS, the chartered institute for IT, has big plans for the future. Our vision is for every Secondary school to have outstanding computing teachers. The BCS scholarship scheme provides you with the opportunity to help inspire students and encourage them into computer science...

STORY BY Abigail Edwards

The last few years have brought about many changes in the curriculum, with a focus on establishing computer science as a foundation subject. This, in turn, has led to more incentives being made available for those who may be considering going into teaching.

Since 2013, BCS has been in partnership with the Department of Education (DfE), offering a teaching scholarship aimed at creating the next

generation of computing teachers for our Secondary schools. The scheme has proved a great success, with more than 1,000 applications over its four years. We've worked with over 130 schools, and with major employers including Microsoft, Google, IBM, Goldman Sachs, BT, HP, Metaswitch Networks, Toshiba, Ocado, Morgan Stanley, and Citrix.

Qualified Teacher Status (QTS) at secondary school level.

Do you want an extra computing teacher in your department for free? Are you able to support a trainee through a School-Centred Initial Teacher Training (SCITT) scheme?

Do you think you have - or someone you know has - the potential to become an

THE SCHOLARSHIP

Eligibility

- You are likely to achieve, or you have already achieved, a 1st, 2.1, or a 2-2 degree.
- Have obtained, or go on to obtain, a place on an ITT course leading to computing/computer science initial QTS at secondary level in England.
- For more information on eligibility, visit: helloworld.cc/2jnGMne.

Testimonial

"Having worked in the software industry for 14 years, I decided to make the switch to teaching due to the exciting opportunity it presents to share my knowledge and enthusiasm for computing and learning. This, coupled with the excellent opportunity presented by the BCS scholarship scheme meant that it was possible for me to make a mid-career switch. I'm now convinced I made the right decision".

- Richard Johnson, BCS Scholar.

Scholarship benefits

- Mentoring
- Training
- Support
- Membership
- Master Teacher Priority

“ARE YOU ABLE TO SUPPORT A TRAINEE THROUGH A SCHOOL-CENTRED INITIAL TEACHER TRAINING (SCITT) SCHEME?”

Apply today

Each scholarship is worth £27,500 (tax-free), and we intend to award around 120 scholarships per year. The funding is supplied by the Department for Education, and is paid in instalments over your year of Initial Teacher Training (ITT). Your course will need to lead to your gaining

effective leader, developing the professional competencies of others? If so, a £27,500 tax-free BCS scholarship is available.

For more information and how to apply, please visit: helloworld.cc/2jqDLCC. (HW)

■ Get involved with computer science education: apps.bcs.org/Scholarship



EXPLORING COMPUTER SCIENCE

ENHANCING CS LEARNING OPPORTUNITIES FOR ALL

What would a curriculum contain if its aim was to broaden participation in CS? Neil Rickus looks at the approach taken by this successful project in Los Angeles, following a visit funded through The Goldsmiths' Company Grants for Teachers.

STORY BY Neil Rickus

Exploring Computer Science is a school and university partnership aiming to “increase and enhance computer science (CS) learning opportunities” within the Los Angeles Unified School District (LAUSD), which has now been adopted by schools in other districts across the USA. ECS also aims to broaden participation in the subject by female, African-American and Latino/Latina students. Student enrolment from all groups has increased year on year since the project began, with students’ understanding of CS and their attitude towards the subject enhanced after participating in ECS. Having been given the opportunity to visit the team and

observe ECS lessons, I was keen to compare the ECS curriculum with the English computing national curriculum (NC). This was particularly motivated by the recent media coverage of Roehampton University’s annual study of computing education, which highlighted the limited number of girls and pupils from poorer backgrounds and ethnic minorities taking GCSE and A-level computing (helloworld.cc/2jmYuY4).

The ECS curriculum can be downloaded for free by schools from exploringcs.org, and is usually studied by 9th-grade students (aged 14–15) with no prior knowledge of CS. The curriculum has a specific topic focus for each half-term, with

individual lesson plans available; teachers can adopt the sessions to the needs of their pupils, for example nancyse.com, which was made by one of the teachers I met. When compared to the computing NC at KS1–3, three areas within the ECS curriculum are particularly noteworthy and could form part of computing lessons in English schools. I’ll look at the pedagogy of ECS in another article.

Computing in society

Despite being named “Human-Computer Interaction”, the initial ECS unit focuses on technology’s broader role in society, rather than specifically on interface design. It’s deemed essential due to the limited exposure to technology within the lives of many ECS students. For example, a teacher at an ECS professional development (PD) event noted that “loads of our students don’t have smartphones”.

The computing NC outlines the importance of using technology “safely” and “respectfully” in KS1–3, with KS3 emphasising the importance of creating projects “meeting the needs of known users” and working with “digital artefacts for a given audience”. While ECS lessons also cover these areas extensively, sessions during the first unit of work spend significant periods examining the appropriateness of technology in a range of situations. For instance, when discussing communication, pupils are asked to consider the implications of being able to interact



■ Collaborative problem-solving is a key component of the ECS curriculum

with several people at the same time, and whether using a text message is suitable for breaking up with a boyfriend or girlfriend. In addition to this, the impact of machine learning and artificial intelligence (AI), including the Turing Test, are explored before working at a computer.

How could I use this in the classroom? Consider discussing the potential impact of pupils' work, such as when programming or producing web-based content. For example, producing a chat bot is a common introductory task in Python at upper KS2/ KS3. Prior to programming, time could be spent examining when and where the technology could be used, how this impacts society, and showing real-world examples, such as the "world's first robot lawyer" at donotpay.co.uk.

Focus on problem-solving

The requirement to analyse and solve problems is a key aim of the computing NC. The concepts and approaches contained within Computing at School's (CAS) Barefoot Computing (barefootcas.org.uk) often form part of pupils' problem-solving

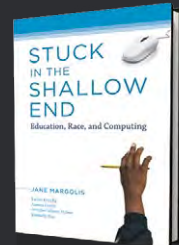
THE RESEARCH BEHIND ECS

ECS builds on the research presented in *Stuck in the Shallow End: Education, Race, and Computing*, which examines the experiences of pupils in three Los Angeles schools. A range of inequalities were discovered to be limiting pupils' opportunities to choose CS as a field of study, with possible solutions proposed. The book was the recipient of the 2009 American Association of Publishers Prose Award in Education.

STUCK IN THE SHALLOW END

BY Rachel Estrella,
Joanna Goode,
Jennifer Jellison
Holme & Kim Nao
(2008)

PUBLISHER MIT Press
PRICE £14.95
ISBN 9780262514040
URL helloworld.cc/2iyBvXZ



noting these problems and concepts are examined before pupils are introduced to a programming environment for the first time.

How could I use this in the classroom? Pupils could be encouraged to articulate their thinking when attempting to solve a problem, with the process formalised depending on the age of the children. It may be appropriate to record/video tasks, which could be subsequently edited by pupils.

uses the relatively inexpensive (US\$33 per device for a class set) Edison Robot meeteditson.com, which is extremely durable and physically compatible with LEGO. The device can be programmed using a block-based programming environment. As part of the ECS unit, pupils enter the RoboCup Junior robotics competition and program the device to "rescue" people.

How could I use this in the classroom? If robots are not available in school, they could be trialled as part of a computing club or digital leaders programme before purchasing a class set. Alternatively, devices could be borrowed from local schools or a CAS Regional Centre where available. When discussing the unit of work, a member of the ECS team noted how engaging the competitions were; pupils could thus be entered into age-appropriate events, such as the recent micro:bit Bloodhound Rocket Car contest or PA Consulting's Raspberry Pi competition.

PUPILS COULD BE ENCOURAGED TO ARTICULATE THEIR THINKING WHEN ATTEMPTING TO SOLVE A PROBLEM

processes, with Computing Unplugged (csunplugged.org) or activities away from the machines used to teach CS concepts.

Within Unit Two of the ECS curriculum, students are introduced to the "four steps of the problem-solving process", as defined by Poyla in *How to Solve It*, which allows them to structure and record their thoughts as they attempt a range of problems. The process is reinforced through examining a range of problems linked to mathematics, including the handshake problem (helloworld.cc/2jADdg0) and the fencepost problem (helloworld.cc/2jABXtb). As with the computing NC at KS3, the binary number system and various searching algorithms are taught. An end-of-unit assignment investigating the travelling salesman problem also forms part of the ECS curriculum. It's worth

Children may also be encouraged to reflect on their behaviour and attitude during the problem-solving process, such as through the rubrics recently published by Phil Bagge on code-it.co.uk.

Engagement through robotics

The need to undertake programming related to physical systems is contained within both KS2 and KS3 of the design and technology NC. Access to devices varies between schools, although the BBC's recent micro:bit project has meant a number of secondary pupils own additional hardware. Primary age-specific devices are increasingly available, particularly at KS2, including the Crumble and CodeBug, and a number of primary schools also make use of micro:bits.

ECS's final unit focuses on robotics and

Evaluating impact

Finally, during the next academic year, the ECS team plan to examine the project's impact over time in more detail, with extensive research findings to follow in due course. In addition to this, an e-textiles unit is soon to be published, which has been well-received in initial trials.

So, why not download the ECS curriculum and try some of the ideas outlined above? Do get in touch on Twitter @computingchamps and let me know how you get on. (HAW)

AGE RANGE

11-12 years

LESSON TYPE

- Visual / block-based coding
- Text-based programming

REQUIREMENTS

- A computer - Linux, Mac OS, or Windows
- Scratch 2.0
- Python 3 with IDLE or access to trinket.io

MOVING FROM SCRATCH TO PYTHON

Many children find the jump from visual programming languages such as Scratch to text-based languages such as Python a challenge. This lesson aims to ease that transition for them...

Visual programming languages are great fun, avoid annoying syntax errors, and are accessible to children of all ages. There comes a time when they must move on, though, and learn to access the greater power and flexibility of text-based languages. This transition can be tricky for

some students, as their tiny syntax errors cause cascades of errors, and misplaced indentation or forgotten semicolons can send their programs spiralling into infinite loops. Here I'll outline one method of introducing a few little snippets of the Python language to students, by using analogous programs in Scratch 2.0.

Getting started

Hopefully, your students will already be familiar with Scratch or some other block-based programming language. You could print the translation grid out or import it into a slide deck, but the key part will be covering up some of the table cells, to provide your students with a challenge.

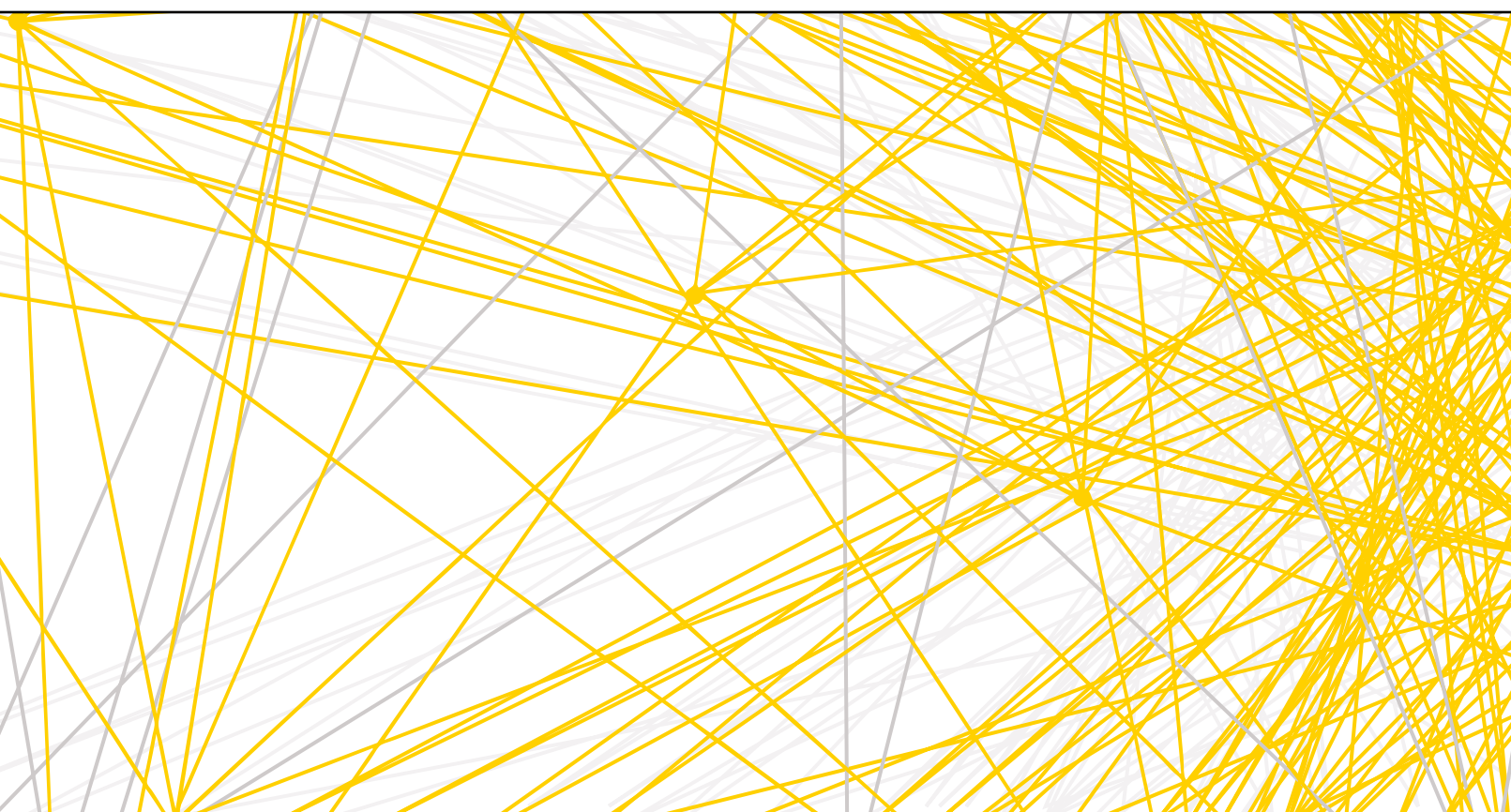
It's a good idea to teach your students how to clear the screen and centre the turtle in both Scratch and Python before you begin.

Initially, you could provide the first row of the table as it is shown here, and let your students try out the scripts in both Scratch and Python, so they can see the similarity of the output and have an understanding of some of the basics of Python syntax.

The next stage would be to cover up the Scratch blocks in the second row. Let your students read through the Python code and try to have them explain what they think the code is doing. They can then try and reproduce this in Scratch.

THE CHALLENGE

- ▶ To begin with, have the students try out some of the scripts in Scratch and Python to see the output produced
- ▶ Next, give your students a small Python script and see if they can reproduce it in Scratch
- ▶ Getting trickier now: get your students to convert a Scratch script into Python
- ▶ Finally, show them a drawing and have them try to reproduce it using both Scratch and Python



Next, you could hide both the Scratch blocks and the output, and give your students the opportunity to try and understand what the Python code is doing,

and have them try and figure out what is missing. They can then use IDLE or [trinket.io](#) to test out the code they have written and try and reproduce the output.

“HOPEFULLY YOUR STUDENTS WILL ALREADY BE FAMILIAR WITH A BLOCK-BASED LANGUAGE

without any hints available. You could continue doing this, or if your students show some aptitude at reading the Python code, you could remove some of the lines of Python

Finally, when your students are ready, provide them with only the output images, have them first write the code in Scratch, and then finally reproduce it in Python. [\(HW\)](#)

ASSESSMENT

- ▶ Which commands are very similar in Scratch and Python?
- ▶ Which commands are different in Python to the ones you use in Scratch?
- ▶ How can you tell when commands are in a loop in Scratch and Python?

ALTERNATIVE ACTIVITY IDEAS

There are other ways you could use this style of activity with students, in a variety of contexts.

7-10 years - Unplugged

- ▶ You might like to provide students with a mixture of Scratch and Python scripts printed out, then have them try to draw the expected output from the scripts.

11-13 years - Physical computing

- ▶ Using NuScratch and the Python GPIO Zero library, you could adapt this lesson to introduce students to physical computing with Python, and show them how much simpler Python can be.

14-15 years - Programming

- ▶ This type of lesson works well when introducing a new language to students. Why not try providing them with comparable Python and JavaScript code snippets?

Resources

Code in editable format:

helloworld.cc/2iDeMND

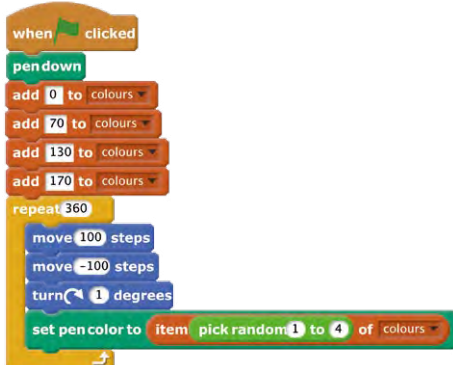
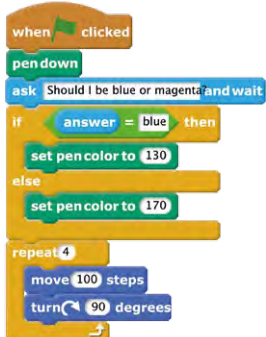
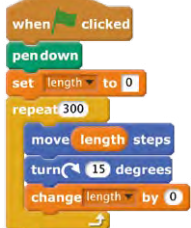
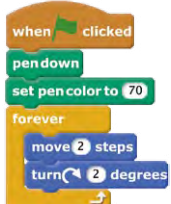
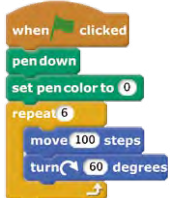
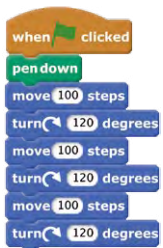
Individual code block images:

helloworld.cc/2iDd0vZ

Python Code in Gist format:

helloworld.cc/2iD8kGx

Scratch



Python

```

from turtle import *
forward(100)
right(120)
forward(100)
right(120)
forward(100)

```

```

from turtle import *
color('red')
for i in range(6):
  forward(100)
  right(60)

```

```

from turtle import *
color('green')
while True:
  forward(2)
  right(2)

```

```

from turtle import *
color('blue')
length = 0
for i in range(300):
  forward(length)
  right(15)
  length = length + 0.1

```

```

from turtle import *
colour = input('Should I be blue or magenta? ')
if colour == 'blue':
  color('blue')
else:
  color('magenta')
for i in range(4):
  forward(100)
  right(90)

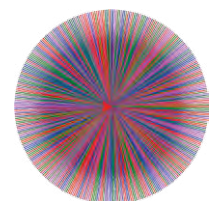
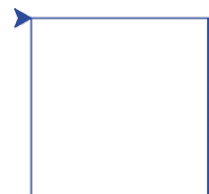
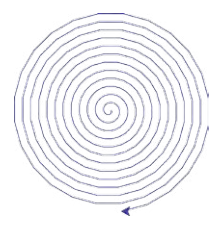
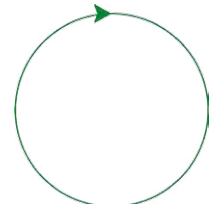
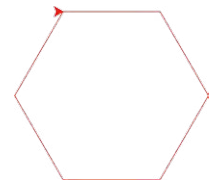
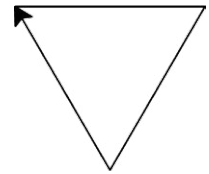
```

```

from turtle import *
from random import *
speed(10)
colours = ['red', 'green', 'blue', 'magenta']
for i in range(360):
  forward(100)
  backward(100)
  right(1)
  color(choice(colours))

```

Output



Apps for Good helps young people learn to create apps that change their world



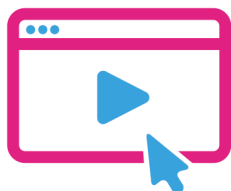
Our course is mapped to the Computing Curriculum and builds skills in teamwork, communication and problem solving



The flexible course framework can be delivered to students aged 10 to 18 and adapted to the needs of your classroom



Our national competition is open to all students, giving them the opportunity to become real-life entrepreneurs



Ongoing CPD helps educators learn up-to-date subject knowledge, technical skills and pedagogy



Students get hands-on experience in tech, business and entrepreneurship careers and the chance to gain work experience



Apps for Good is free for all non-fee paying UK schools and is a great way to showcase your school's ethos in action

"Never has an educational programme opened up so many opportunities to students and so many doors to industry."

Chris Aitken, Head of Computing Science, Scotland

To join as an Education Partner visit:
appsforgood.org



AGE RANGE

16-18 years

LESSON TYPE

Unplugged

REQUIREMENTS

- Pen and paper

JUNGLE MAZE SOLVER

Introduce students to pathfinding algorithms and build their problem-solving skills with this jungle maze scenario

You're on an expedition in the middle of the deep jungle; it's very thick and overgrown. A party of fellow explorers have become lost and sent back a distress

signal with their coordinates. You're safe at camp and need to come up with a plan to bring them home safely, but the jungle is very dangerous and the camp

leader doesn't want to send any more people in case they get lost too. Luckily, at camp you have access to a map grid and a programmable robot.



THE CHALLENGE

- Some of the map squares are clear; in some squares, prickly plants block the path
- The robot can detect whether the map square immediately ahead is clear or not
- The robot can move forwards, reverse, and rotate left/right
- The robot isn't remote-controlled - once you release it, it's gone!
- The robot has a memory
- The robot can be programmed with rules to follow. An example of a rule might be "If the square ahead is blocked, rotate 90 degrees right"

OPTIONAL

It's getting dark - the quicker you find the lost explorers, the better!

Devise a set of rules for the robot to follow to find the explorers

Rather than simply finding a path through one maze, this lesson focuses on creating an algorithm which can be generalised to solve any maze it's presented with. The aim is for learners to work together to think about how this could be achieved, before demonstrating their algorithm to the class. Ideally, after presentation there should be opportunity for reflection and, potentially, further iteration upon their solution.

As an extension, learners could swap strategies with another group and attempt to follow the other group's rules. Are the rules they provided clear? Different groups could test different mazes to

investigate whether the design of the maze makes any difference to their solution. Groups could devise mazes they think are difficult and set them as a challenge for other groups. What properties would make a maze difficult to solve? Would it make any difference if you could see ahead by more than one square? Is there anything else you would want the robot to be able to do that it can't currently do?

ASSESSMENT

- How did you discover bugs in your instructions for the robot?
- Describe a bug and how you solved it
- How could your instructions for the robot be improved?

ALTERNATIVE ACTIVITY IDEAS

Explore and make

Adapt this project theme for other age groups:

5-6 years - Unplugged

- Ask students to program Bee-Bots to reach the lost explorers
- Construct a real-world maze where students program each other

11-13 years - Programming

- Trace a path to the lost explorers using turtle software
- Write a program to auto-generate a randomised jungle maze in Minecraft

14-18 years - Physical computing

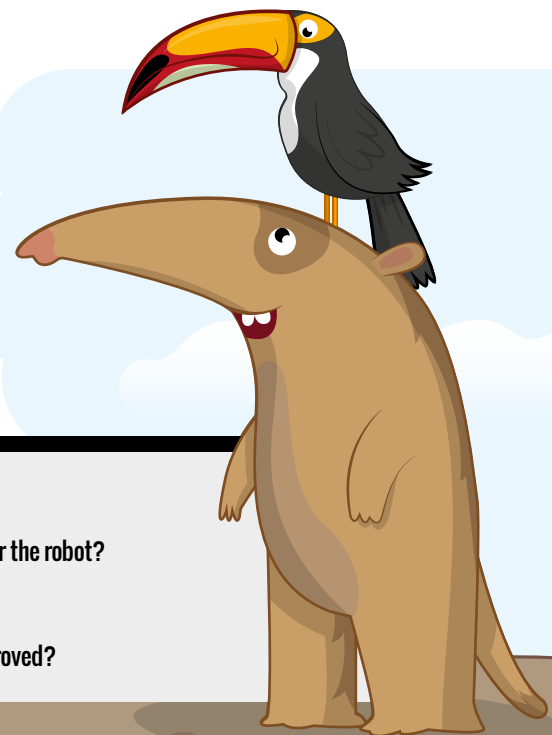
- Build a robot and program it to move through a physical maze

FURTHER READING

Maze solving: helloworld.cc/2k4zUul

Basic Raspberry Pi robotics: helloworld.cc/2iNFY9p

The A* algorithm and Dijkstra's algorithm are included in A Level Computer Science; a natural progression from this lesson would be to examine and test these algorithms on the jungle maze. (HW)



AGE RANGE

14-16 years

LESSON TYPE

Programming

REQUIREMENTS

- Pen and paper
- Python 3

ESCAPE FROM RAVENSWOOD MANOR

Build a text adventure with a twist – introducing the dictionary data structure.

It's a dark and chilly night at Ravenswood Manor. You awake in an unfamiliar room with little idea of how you got there, and

an overwhelming urge to escape. Will you be able to find the path to safety, or will you fall prey to Lady Ravenswood and her minions?



THE CHALLENGE

- 👤 Design your layout on paper
- 👤 Devise some fiendish traps for the player to encounter
- 👤 Decide the routes through the manor and the location of the exit

Students begin by designing the inside of Ravenswood Manor on paper. Often, the task of planning before beginning to program is an unpopular one, but for this particular exercise students will find their planning crucial to the success of the task.

The aim is to introduce the dictionary data structure, so that we can model the rooms of the manor and where the player can move in between rooms.

Create a dictionary for each possible direction a player could move. For example, in a 3x3 grid:

```
north = { 0: None, 1: None,
          2: None, 3: 0, 4: 1, 5: 2,
          6: 3, 7: 4, 8: 5 }
```

The format 0 : None means when in room 0 and they choose north, move to None (i.e. there is no path to the North). Or, 4:1 means when in room 4 and they choose north, move to room 1.

We can also use dictionaries to store the descriptions of the rooms:

```
description = { 0: "You are
in a cold damp cellar. Rats
are everywhere.",
               1: "You are in a kitchen
```

```
store room",
               2: "You are in a long
chilly corridor" }
```

Traps in the rooms...

```
trap = {0: "You spot some
cheese in a corner. Should
you pick it up? y/n" ...}
```

...correct answers to the traps...

```
correct_answer = { 0: "n"
... }
```

...and responses, depending on what was chosen...

```
trap_y = { 0 : "Thank
goodness you didn't pick up
the cheese, it is infested
with rats" ...}
trap_n = { 0 : "You get set
upon by a swarm of rats and
die horribly. Oops." ...}
```

Students can then use programming concepts they are familiar with to piece together the adventure:

```
current_room = 0
where_next = ""
while current_room is not
None:
```

RESOURCES

Dictionaries in Python: helloworld.cc/2jyHWhR

Download the code: helloworld.cc/2k4cfht

ALTERNATIVE ACTIVITY IDEAS

Explore and make

Adapt this project theme for other age groups:

11-13 years - Programming

- 👤 Provide a skeleton program with the dictionaries already initialised

16-18 years - Programming

- 👤 Add to the program to allow the player to pick up and interact with items
- 👤 Add hit points - instead of causing instant death, traps can damage the player
- 👤 Add a sound or a picture to display upon entering each room

```
print(description[current_
room])
where_next = ("Which
direction? ")
if compass[where_next]
[current_room] is not None:
    current_room =
compass[where_next][current_
room]
else:
    print("There is no path")
```

(HW)

ASSESSMENT

- 👤 How did planning your game help you?
- 👤 Why are dictionaries more suitable than lists for this task?
- 👤 Did you include any input validation? Why is validation important?

AGE RANGE

7-10 years

LESSON TYPE

Visual / block-based coding

REQUIREMENTS

- Scratch 2.0

MY AMAZING CASTLE

Create a castle with a dragon that performs actions when it bumps into objects. Maybe your dragon will change colour when it flies over a rainbow...

Create a project that allows the player to control the dragon by placing objects around the stage. Start with the rainbow and candle, and then add your own objects.

Lesson Outline:

- Demonstrate the completed project on an interactive whiteboard.
- Step through the code, explaining any concepts that are new to your class.
- Give children the instructions and sample code, and have them complete the project.
- Share projects with the class or give pupils the chance to try their projects out on younger children in the school.

THE CHALLENGE

- Create a new Scratch project and delete the cat sprite.
- Add the castle4 backdrop.
- Add a Rainbow sprite and its code. Shrink the sprite. Turn on the 'can drag in player' setting, so that the player can drag Rainbows around, even in Fullscreen mode.
- Add a Candle sprite and its code. The Candle has the same code as the Rainbow; you can drag the code from the Rainbow to the Candle sprite to copy it and save work.
- Add a Dragon sprite and its code.
- Add more objects for the dragon to react to. What might the Dragon do if it bumped into a rock or found a microphone? What if the Dragon met a lion? Drag the Rainbow to the Candle sprite to copy it and save work.



Scratch
Create
Explore
Discuss
About
Help
Search
Join Scratch
Sign in

My Castle Example

remixed by CodeClubRik

13 scripts
7 sprites
See inside

Instructions

Click the green flag to start.

Click on an object to make a copy and then drag the copy to where you want it. When the dragon touches an object it performs a corresponding.

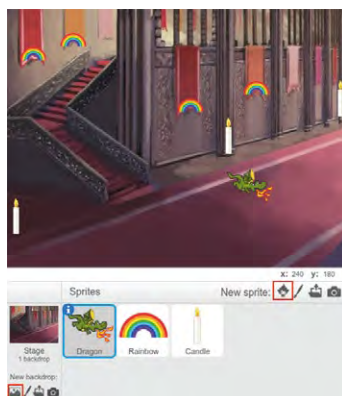
Notes and Credits (added by CodeClubRik)

Influenced by Seymour Papert's My Make Believe Castle

Thanks to drgardner for: My Castle (Original Project)

© Shared: 15 Dec 2016 Modified: 15 Dec 2016

0
0
3
2



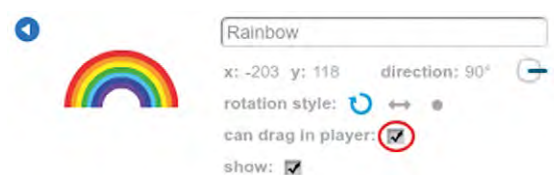
It's important for children to learn how to create a project that will be used by someone else. When KS2 pupils have made their project, pair them with younger children in the school. The children who have developed the projects should explain how to use

Click the green flag to get the dragon flying around. You can drag objects (rainbow and candle) around the stage. Click on an object to create a copy of it. When the dragon flies over an object, it will perform a corresponding action. ▲

the project, and then let the younger pupils place icons to customise the scene. This will enable the KS2 pupils to learn how to create a project with a user in mind, and explain how their project works.

Inspired by Papert

This project was inspired by the work of Seymour Papert. Papert realised that young children could learn to code if they were given the right tools. Papert's My Make Believe Castle allows children to customise a castle by placing icons that trigger behaviours in characters. (HW)



ALTERNATIVE ACTIVITY IDEAS

This lesson can be adapted to different age groups.

5-6 years - Unplugged

Children create their own castle scene, using a finished project created by older children or a teacher. They learn that they can control what the dragon does by placing pictures.

11-13 years - Programming

Learners extend the idea by having characters interact with other characters. Add multiple 'rooms' to the castle with different objects and characters in each.

ASSESSMENT

- Did you learn any new Scratch skills in this project?
- What do you need to do to add a new object and action for the dragon?
- How could you make your project more fun for younger children?

FURTHER READING

Example project: helloworld.cc/2jnq73c

My Make Believe Castle: helloworld.cc/2jnFUPT

AGE RANGE

11 - 13 years

LESSON TYPE

Text-based
programming

REQUIREMENTS

- groklearning.com website (free to access)
- Computer
- Internet connection

PYTHON TURTLE FLAGS OF THE WORLD

Program the turtle to draw flags from around the world, and design your very own flag, in this introduction to programming in Python via groklearning.com

M eet the turtle. It can draw all sorts of things... with a little help from you!

Write your own programs to help the turtle draw and colour flags from around the world, or get creative and

design your very own flag.

This beginner-friendly online coding activity introduces you to the programming language Python. You will write and debug your own code, develop computational

thinking skills, and get a little geometry practice along the way. The turtle makes programming visual, allowing you to see exactly what your program is doing at every step.

An example problem from the Flags of the World activity. Circles represent the interactive note slides; diamonds represent the problems

The editor where students can write, test, and debug their programs. Running the program shows an animation of the program in action below

Students can submit their program for auto-marking to check that it's correct, and to see hints for fixing it if it's not.

The screenshot displays the Grok Learning interface for the 'Flags of the World' activity. On the left, the 'Flag of Germany' task is shown with instructions: 'Let's draw the flag of Germany! Write a program to draw a flag that is 180 turtle steps wide, and 120 turtle steps high. The top third should be 'black'. The middle third should be 'red'. The bottom third should be 'yellow'. When it's finished, the result should look like this: [Image of the German flag]'. On the right, the Python code editor shows the following code:

```
1 from turtle import *
2 color("black")
3 begin_fill()
4 forward(180)
5 right(90)
6 forward(40)
7 right(90)
8 forward(180)
9 end_fill()
10
11 right(180)
12
13 color("red")
14 begin_fill()
15 forward(180)
16 right(90)
```

Below the code editor, there are buttons for 'Run', 'Terminal', 'Save', 'Copy', and a red button labeled '★ Mark Flag of Germany'. A preview window at the bottom right shows the flag being drawn by the turtle.

THE CHALLENGE

- 🚩 Read the interactive notes and solve the coding challenges
- 🚩 Program the turtle to draw flags from around the world
- 🚩 The turtle follows a sequence of instructions to draw on-screen
- 🚩 The turtle can move forwards or backwards, and rotate left or right
- 🚩 Move the turtle by specifying the number of turtle steps
- 🚩 Turn the turtle by specifying the angle of the turn
- 🚩 Use Fill to colour shapes the turtle has drawn
- 🚩 Submit your program for auto-marking and find out if it's correct!

This online activity is structured as a sequence of interactive notes and challenge problems for students to solve. As students learn how to manipulate the turtle, changing angles and drawing lines and shapes, they are challenged to draw flags of increasing complexity.

Interactive notes

Most note slides contain interactive examples which can be run by clicking the ► button in the top-right hand corner of the example box. Encourage students to run these examples when they see them.

Challenge problems

After writing, running, and debugging their program, students can submit it to the auto-marker to check if it's correct and, if not, to see hints for fixing it.

If students run into difficulties solving the problems, encourage

them to try to persevere with them independently. They can look at the marking notes for hints to help them diagnose what's wrong, or go back through the content slides. As a teacher, you also have access to Teacher's Notes in the top-right corner of the header, where you will find explanations of the solutions.

Creative play

At the end of the activity there is a Playground where students can further experiment with the skills they have been practicing. Seymour Papert first introduced turtle graphics with the aim of giving students open opportunities to create, discover, and extend their own learning. Now that your students have mastered the basic turtle commands, the Playground offers them this opportunity for experimentation and creative play. (HW)

ALTERNATIVE ACTIVITY IDEAS

Differentiation Ideas

This activity can be adapted to provide more scaffolding and support for younger and less confident students, or extended for older students.

7-10 years - Unplugged

- 🚩 Introduce younger students to turtle programming by having them physically act out turtle commands
- 🚩 Have students work in groups to write, test, and debug a set of instructions which one student, the turtle, then has to follow
- 🚩 Challenges might be to walk once around the whole classroom, starting and finishing at the door, or to make it through an obstacle course

11-13 years - Collaborative programming

- 🚩 Work through the content slides and interactive examples as a class. Encourage students to make hypotheses about the examples before running them
- 🚩 Have students solve the problems together as a class or in small groups. Encourage peer mentoring for students who are stuck
- 🚩 If students are pair programming, have the less confident student do the typing

14-16 years - Text-based programming

- 🚩 Encourage students to come up with their own flag designs and run a competition to pick the best design
- 🚩 Challenge students to draw the flag of their own country, or that of their parents or grandparents
- 🚩 Introduce students to more concepts, including loops, by moving onto the Frozen Fractals activity

FURTHER READING

What is Python?: www.python.org

Papert's Turtle: helloworld.cc/2iyr5Yu

ASSESSMENT

- 🚩 What commands did you learn to program the turtle?
- 🚩 How did you decipher error messages and/or fix bugs?
- 🚩 How did you work out the angle you needed to turn?

STORY BY Emma Goto IMAGES BY Yoshihiro Goto

PLAYFUL COMPUTING

How do we get young children thinking computationally and taking their first steps with programming? Emma Goto, Senior Lecturer in Teaching Development at the University of Winchester, believes the key is to let them play...

Experienced teachers of young children will tell you that children learn an enormous amount through their play. Before children start school, the list of what they have already learned is extensive. In most cases, they learned these early skills not through direct teaching, but through play and exploration motivated by curiosity and wonder. If we want to develop young children's computational thinking and understanding of computing, we should capitalise on these experiences.

Jeanette Wing introduced the term 'computational thinking' to describe approaches to problem-solving that utilise key concepts from computing. Approaches such as logical thinking, decomposition, abstraction, generalisation, and the creation of algorithms are the basics of computational thinking.

Those involved in the education of young children understandably have concerns about the risks of too much screen time. However, many of the richest opportunities to develop children's computational thinking don't even involve digital technology. In the Early Childhood setting, children are playing all the time. They construct their understanding of the world through their play. Whether they are building a den, making Lego structures, doing a jigsaw, drawing a picture, or playing with puppets, children will encounter problems in their play. Problems may include how to move some water to the digging area, or how to build a bridge across the car track, and can provide opportunities to develop children's computational thinking. These problems belong to the children, therefore they are meaningful to them, and the children are very motivated to solve them. In these situations, the Early Childhood Practitioner can support

the development of children's thinking both through their interactions and through careful questioning.

By asking children if they have ever seen a similar problem, we are encouraging them to identify patterns and we can support them to generalise prior solutions. We promote abstraction when we ask children if they can draw or build a model of their problem. When we ask children to break the problem up, sharing out responsibility for different tasks, we are nurturing the skill of decomposition. By encouraging children to talk through solutions, we are supporting them to develop an algorithm. Questions that urge children to consider 'What will happen if...?' encourage logical thinking. Many practical play situations provide opportunities to get children thinking computationally, without going anywhere near a computer or digital device.

Programmable toys

Playful approaches can also be used to support children to take their first steps in programming: Bee-Bot, Blue-Bot, and Roamer-Too are just a selection of the range of programmable toys available nowadays. These programmable toys enable children to create simple programs in a fun and practical way. Let's look now at how children's early programming skills can be developed using the Bee-Bot (alternative programmable toys could, of course, be used).

The Bee-Bot is a small toy which is programmed using seven buttons on the top. Each of the buttons corresponds to

first introduced, children will need time to play freely with it. This will allow them time to explore what each of the buttons do and develop an understanding of how the Bee-Bot behaves. Once children have familiarised themselves with the Bee-Bot through free play, they are ready to move on to solve more structured problems with the toy.

One way to facilitate this is through the use of a grid or mat; it's possible to buy mats for most programmable toys. Alternatively, children and practitioners can create their own mats. This gives children ownership of the mat and also allows them to be tailored to match the stories, themes,

Children can collaborate to solve problems with programmable toys. Giving children individual whiteboards, and asking them to plan out their algorithm before programming the toy, supports the development of computational thinking.



thinking through this type of activity, ensure that children are given time to produce an algorithm before they create their program. Give them a small whiteboard and dry-wipe marker so that they can write or draw their algorithm. This will be a simple plan of the steps they will need to go through to solve their problem. Once they have this algorithm, they can use it to program the toy.

“ ENSURE THAT CHILDREN ARE GIVEN TIME TO PRODUCE AN ALGORITHM BEFORE THEY CREATE THEIR PROGRAM

a command; therefore, the Bee-Bot is programmed using a basic programming language consisting of only seven commands. There are buttons to make the Bee-Bot move forwards or backwards, buttons to make it turn left or right, a button to make it pause, one to clear the program, and a 'Go' button to run the program. The small number of commands means the Bee-Bot is simple to use and understand for young learners. When the Bee-Bot is

or topics that are being followed within the curriculum at that time. Bee-Bots travel in steps that are 15 centimetres long; therefore, a simple Bee-Bot mat can be created by drawing out a grid of horizontal and vertical lines 15cms apart on a 60cm square of paper, creating a 4x4 grid. In the photograph here you will see an example of a mat created around the theme of the book *The Very Hungry Caterpillar*.

To develop children's computational

Making progress

Teachers often worry that if children use the same programmable toys over a number of years, there will be no progression. However, progression does not come from using more complicated equipment; instead, we need to provide more challenging problems to ensure the children make progress with their learning. For example, by using the Bee-Bot mat that was illustrated earlier, children new to programming may be asked to move the Bee-Bot to the apple. The more experienced programmer might be set the challenge of moving the Bee-Bot to each of the items of fruit the caterpillar ate in the correct order, whereas children who are more capable might be required to move the Bee-Bot to each piece of fruit eaten in the correct order, pausing on each meal in turn whilst avoiding going over the same piece of fruit twice and steering clear of the junk food. If children are able to solve the problem first time, without the need to debug their solution, the problem posed was too simple. The Bee-Bot will allow you to create a program with up to 40 commands, so there is scope to create some longer programs to solve more complicated problems.

So, if you want to get young children creating programs, encourage them to use playful approaches. (HW)

OTHER USEFUL RESOURCES

Programmable toy rulers - Simple rulers the length of one step of the programmable toy can be produced to support children when solving problems. Bee-Bot moves in 15cm steps, so 15cm Bee-Bot rulers can be created.

Instruction cards - Sets of instruction cards can allow children to represent their algorithm. These cards can be reordered easily, supporting children to debug their solutions. These types of instruction cards can often be purchased from programmable toy manufacturers but home-made cards are also easy to produce.

Mazes - You can buy various mazes and maps for different brands of programmable toy; however, these can also be created using resources such as wooden building blocks.

Route cards - Square cards, with sides the same length as one step of the programmable toy, can be produced to mark out routes that children program the toy to follow. Bee-Bot moves in 15cm steps, so square route cards with 15cm sides should be used.



TURTLE GRAPHICS MADE EASY IN SCRATCH

Easy to adapt to the abilities of your pupils, lots of space for exploration in a maths-based context, and great opportunities for computational thinking. Here are three activities you could try today...

Turtle graphics can be used with pupils as young as seven. Pupils' programming skills can be developed alongside their mathematical knowledge and understanding. For example, pupils can use turtle graphics to explore simple right-angled shapes, discover the properties of regular 2D shapes, create and adapt repeated patterns, draw shapes using coordinates, and write programs to translate, rotate, and enlarge shapes.

Exploring right angles

Mark out stairs, squares, rectangles, and block letters (such as H, E, and T which can be drawn using right angles) on the playground or on the classroom floor using chalk or masking tape. Ask the pupils to walk on the shapes, whilst recording the steps they need to create the shapes using Scratch language on a whiteboard or paper (move x steps, turn right/left 90 degrees). As pupils record the route they take, they are recording their algorithm. As they then

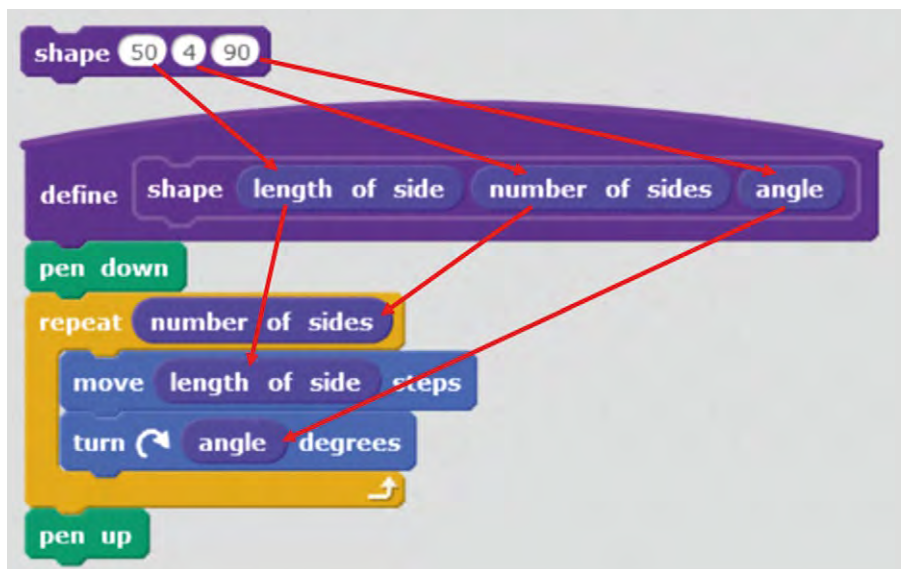


code their instructions in Scratch, they are converting their algorithm into code. You will need to explain that Scratch steps refer to pixels (dots) on the screen, while human steps are much larger. Multiplying one human step by ten when on Scratch works well. If their code has a bug (error), they can go back over their physical shape to see what part of the algorithm is incorrect. Reserve your highest praise for pupils who debug errors: in doing so, you are encouraging them to become independent, problem-solving learners.

Properties of regular 2D shapes

Challenging pupils to draw regular 2D shapes using Scratch is a great way to bring a maths lesson alive. Don't forget to draw a shape other than a square on the floor, and use elicitation to emphasise that pupils are using part of the exterior angle, rather than the interior angle. You could also ask pupils to work out the interior angle and leave it as a comment attached to the code. Ask pupils to create their shape programs using the smallest amount of code. In doing this, you're encouraging them to think about algorithmic elegance and efficiency. At a later stage, this can translate into how much processing power or storage a program uses, but in the early stages it can just be about using the least code. This is a great opportunity to introduce repetition through the 'repeat x times' loop, if pupils haven't already discovered it.





MORE DETAILED PLANNING

Barefoot computing has more detailed 2D shape planning at barefootcas.org.uk. The code-it site has sheets to help assess 2D shape patterns (helloworld.cc/2ijvHS7) and investigate coordinates (helloworld.cc/2ijzTA), as well as investigating translation (helloworld.cc/2ijxVkt), enlargement, and rotation (helloworld.cc/2ijxVkt) using turtle graphics in Scratch.

2D shape patterns

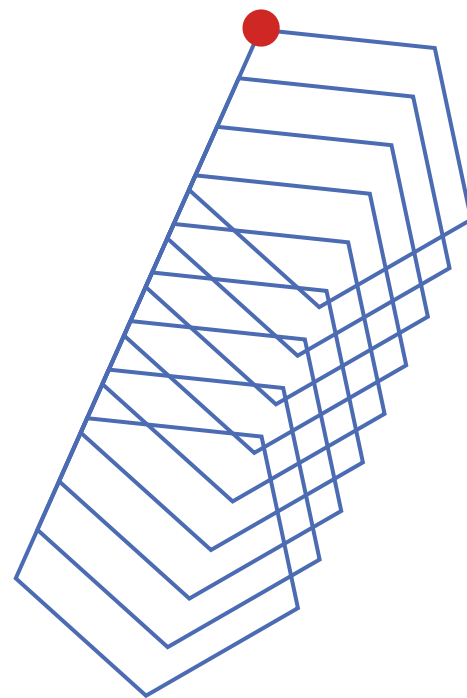
If pupils haven't already discovered it for themselves, show them how a shape can be repeated by nesting the shape code within another 'repeat' loop. Allow pupils plenty of time to explore the patterns they can make using a wide variety of 2D shapes. Good teacher prompts are vital here. How can you make that shape rotate all the way round? Could you make the gap in the middle smaller or larger? How can you fit all your shape on the screen? After they have exhausted this creative play, explain that computer scientists have a way of writing one section of code that can be used to draw any regular 2D shape. Explain that the same code can draw an equilateral triangle, a square, or a regular pentagon. Ask them what properties all regular 2D shapes have. They can look at their code examples to check they all have length of sides, number of sides, and angle of turn. If we were asking a human to draw a regular 2D shape, we could tell them those three things and they could draw the shape. Using the 'more blocks' tool in Scratch 2.0, create a block called 'shape' with three numerical inputs. Add those inputs called 'length of side', 'number of sides', and 'angle' into the move, repeat, and angle blocks as illustrated. Pupils can now use this procedure block to create any regular 2D shape. Allow time to experiment with this idea.

Assessing understanding

With all turtle graphics programming it can be hard to assess pupils' deep understanding from work output alone. It's possible to create complex patterns without a full understanding of what is happening in the code. A really useful assessment activity is to present pupils with a series of progressively more complicated patterns for which they need to write an algorithm before they try to replicate the code. Their algorithms are a best endeavour in a short time frame, but they reveal lots about pupils' comprehension of the task. Do they use the procedure blocks? Do they understand the importance of the loop? If they're not used to creating algorithms, then an example can help introduce a basic pseudocode.

```
Repeat 9
  shape 50 5 72
  move 20
```

Solving problems using 'move', 'turn', and 'pen' turtle command blocks in Scratch is something that every pupil should experience in their primary/elementary education before moving on to work with coordinates, drawing, translating, enlarging, and rotating shapes, and investigating even more complex spatial puzzles through the wonderful medium of turtle graphics in Scratch. Pupils can extend their understanding further into 3D design



using Beetle Blocks (beetleblocks.com/run), and everything learnt easily translates into text-based programming in languages such as Python, which has its own turtle graphics library. (HW)



Phil Bagge
Computing Inspector/Advisor
Hampshire, UK
[@baggiepr](https://twitter.com/baggiepr)

WRITTEN BY Monique Dewanchand, Peter Kemp and Tom Haines

GET STARTED WITH FREE 3D MODELLING SOFTWARE

Have you ever wanted to make VFX and games graphics like the professionals? Blender allows you to do this for free, from the comfort of your own computer...

WHAT YOU NEED

○ Blender - blender.org/download ○ Mac or Windows / Linux PC

3 D graphics are all around us, in the games we play and the TV and films we watch, yet very few people know how to make them. 3Dami and b3d101 allow 6- to 18-year-olds to express their creativity in three dimensions, using industry-standard tools. 3D animation can also be a great way to teach some core computational thinking concepts like:

Decomposition - Breaking a film down into shots, models, animations, sets, lighting, etc. Breaking down models into materials, textures, bones, faces, vertices, etc. Breaking down animations by first blocking it out, then adding extremes, and finally polishing.

Pattern recognition - Using base models to build different characters, vehicles, etc. which share common attributes. Recognising the patterns inherent to creating realistic animations. Sharing assets across multiple shots.

Abstraction - Visual abstraction: the reduction in detail to reduce the render time of poorly observed objects; this may involve only modelling part of a scene or making a 'low-poly' representation of an object that doesn't feature prominently in a scene. Artistic abstraction, where an object is represented in a deliberately unrealistic style, with students recognising the features critical to the representation of the object.

Algorithm - The process of making a short film, from storyboarding to using keyframing for animation. Techniques for making specific

effects, e.g. the 'recipe' for making a virtual building explode so that it looks realistic. 3D graphics theory as it applies to technical artists, e.g. optimising render time.

In this tutorial you will learn to make a snowman in Blender, a free tool used by the industry. This will teach you the basics of 3D animation: how to add, delete, move, rotate, and resize objects, and finally render (output) your masterpiece.

STEP 01 Download Blender

Blender is free; it works on Windows, Mac, and Linux, and really old hardware. You can even run it off a USB stick or a shared drive.

STEP 02 Run Blender

When you run Blender you will meet a splash screen like the one below; click to the right to get rid of it.

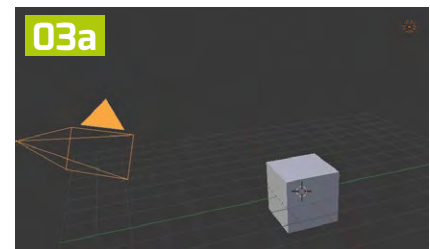


NOTE: RIGHT SELECTS, LEFT ACTS

Unless you play Starcraft (a RTS computer game) it might seem odd that in Blender the right mouse button selects, whilst the left mouse button acts. This may slow you down as a beginner, but experienced users work much faster as a result of this design decision. Without selection/action ambiguity, much fewer mode switches are required to perform common operations. If you're unconvinced, go watch a professional Starcraft tournament!

STEP 03 The snowman's body

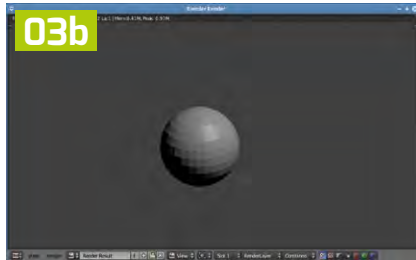
Blender starts with a simple film studio setup, that contains a camera, one light, and a single prop in the form of a big grey cube:



Snowmen aren't made of cubes, so we need to remove it. Make sure you've selected the cube by right-clicking on it, then press the X key and press Enter to delete it. You can see which object is selected as it has an orange border.

Now we need to add a sphere for the snowman's body. On the left-hand side of the screen, click on the Create tab, then click on UV Sphere. A ball will appear where the cube was.

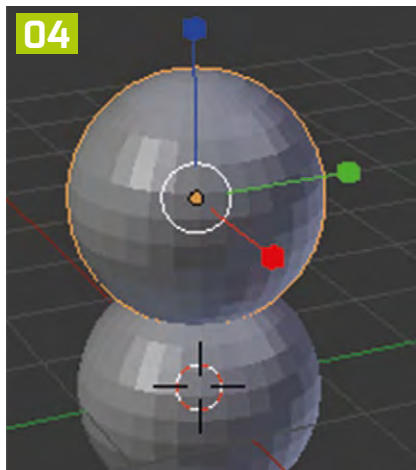
Press F12 (Fn-F12 on a Mac) to render the image you have created thus far, to see what it looks like. Press Esc to get back to the previous view. Don't worry if you can't see your sphere yet; keep reading!



STEP 04

The snowman's head

Add another UV sphere using the Create tab; this will be the head. You will probably find that the sphere appears on top of your previous sphere, so will need to move it. Right mouse click on the sphere to make sure it's selected; three arrows should appear:



(If the arrows don't appear, at the bottom of the screen find the 'move' arrow button and select it).

Left mouse click on the blue arrow to move the UV sphere vertically, and place it above the body.

The snowman's head is a little too large for its body, so resize it by selecting the head (remember to use the right mouse button), then clicking on the Resize button.

Left mouse click on the square handles and move them to reshape the snowman's head. (The more experienced amongst you might want to press the S key and move your mouse instead).

NOTE: THREE-BUTTON MOUSE

Blender is easiest to learn with a separate three-button mouse. For those of you using a laptop, you can emulate the third mouse button by going to File -> User Preferences... -> Input -> Emulate 3-Button Mouse. You can now hold Left-Alt then press the left mouse button to rotate. If you are using a trackpad you might also be able to zoom in and out by pinching (OS X) or two-finger swiping

At this point you might want to check that the head is in the right location and is the right shape. To move around the 3D space you can zoom in and out by scrolling your mouse wheel, and rotate the view of the snowman by pressing the middle mouse button and moving your mouse.

STEP 05

The snowman's nose

A snowman can't smell without a nose, so we need to create a carrot in Blender. First, add a cone and move it to the front of the snowman's face; you'll also need to make it small enough to fit (see above for how to move and resize). We now need to get the nose to point in the correct direction; first, select the Rotate button.

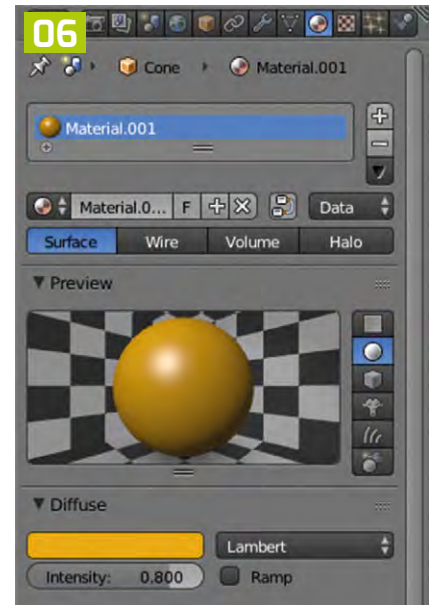
The coloured arcs are handles which we can use to rotate the cone. (Experienced users: try pressing r and x, y, or z to get the same effect.)



STEP 06

Colouring things in

You might have noticed that the carrot nose needs to look a little more carrot-like. To colour it in select it (right mouse button again). Now click on the Material tab on the right-hand side, then click New and click on the Diffuse Colour swatch to make the nose orange..



STEP 07

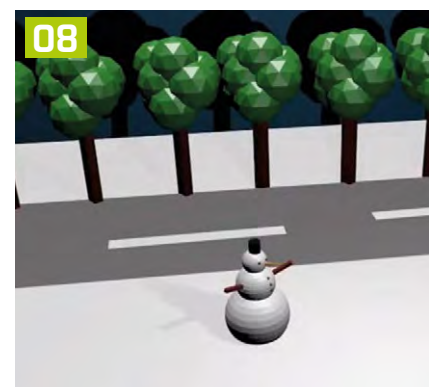
Rendering your masterpiece

Press F12 (Fn-F12 for Mac) to see your masterpiece. If the camera misses the snowman, use the middle mouse button to rotate around your scene and find out where your snowman and camera are. Remember that you can move and rotate the snowman to get it in shot, or even rotate and move the camera.

STEP 08

Next steps

If you have the time then add some eyes, buttons, arms, and a hat to your snowman. Then, to create a complete image, add the ground (try creating a plane), some trees (UV spheres on a cylinder) then, finally, tweet your snowman with the #b3d101 and #3Dami hashtags. (IHW)



AFTER SCRATCH TRY SNAP!

Snap! is an open-source blocks-based language that runs in the browser. It's an ideal progression from Scratch, useful up to sixth form and beyond...

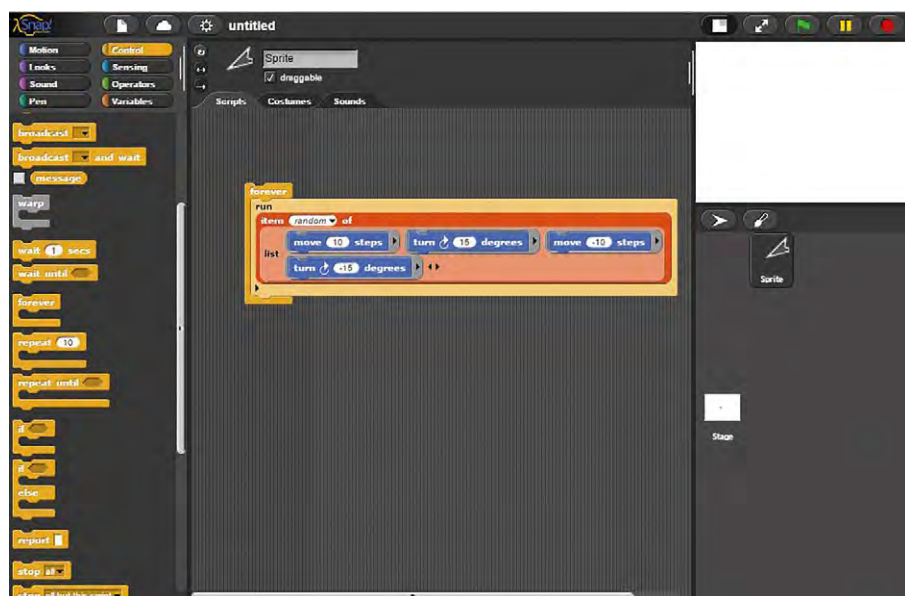
STORY BY John Stout

Starting Snap! (from the University of Berkeley: snap.berkeley.edu) after Scratch won't cause your students any problems, since the interface and visual metaphors are identical to Scratch. Look at the Snap! screen image and while you're there, try and guess what the code shown does; the answer is below. Snap! borrows a number of features from Scheme (see 'Scheme disguised as Scratch' below) which give it far more power, and the interface has a number of additional features which let students accomplish tasks more easily and quickly.

Building Your Own Blocks

BYOB (Build Your Own Blocks) was the name of Snap!'s predecessor, and gives the first hint of what differentiates it from Scratch.

Right-clicking anywhere in a script area brings up a menu with the magic words 'make a block', and then a dialogue to select



The Snap! screen won't worry any student used to Scratch, but be ready to show the differences

SCHEME

If you haven't seen Scheme, it's worth having a look at this descendant of Lisp: it has simple syntax, but remarkable power. The Racket system (racket-lang.org) and Bootstrap (bootstrapworld.org) are both based on Scheme-like languages, and there's a lot of educational material available. See also mitpress.mit.edu/sicp/ for details on the book Brian Harvey (Snap! designer) calls 'the Bible'.

what category your block should be in, its name, and the type of block: a command (procedure), a reporter (function), or a predicate (function that returns a Boolean).

I find the best way of naming a block is to write a sentence describing it. So in the example below, which deletes all occurrences of a value from a list, returning a copy of the list without the values specified, I've used almost exactly that (see 'Defining a block'), defined it as a reporter since it returns something, and put it in the Lists category.

You then specify which of the words in the name are Input names (parameters) and which are Title text (part of the name) by clicking on the word; the

parameters can appear before or in-between parts of the name, not just after as in most text-based languages. Next, you give their type; 'value' could be any type, but 'list' must be a list (see 'Defining block parameters in Snap!'). Finally, you get to define the blocks that, when executed, accomplish what your block needs to do.

In the example (see 'Creating the code for the block in Snap!'), this is defined as a recursive function as no loops or variables are needed; since it calls itself, as soon as you have defined the heading (name and parameters) you need to click Apply, so that you can drag the updated definition out of the relevant category.



■ Click on the word that will be a parameter, change it to an input name, and specify its type.

Useful tips:

- You can use the keyboard editing (see next section) to find a block; this is often a quick way of finding one particular block.
- Right-clicking a block often gives you the chance to relabel i.e. change it without having to delete it and then insert the correct definition.
- If you're in the middle of defining a new block and realise that you need to make another new block, you can keep that definition active, right-click on a blank area within it, and select 'make a block'.
- In the Variables category there's a block called [script variables (a)], which lets you create one or more variables with a scope limited to that particular script. Click the variable name to rename it, and click to add a new variable and remove the last one.

Keyboard entry, tables, debugging

It's often suggested that the drag-and-drop interface holds students back, so one of the many improvements made to the interface in Snap! lets you use the keyboard to enter your programs.

First, make sure the feature is enabled by clicking on the Tools menu (the cog icon) and then Keyboard Editing; enable the Table support and Visible stepping as well, since we'll look at those later. If you can't see any or all of these, you may not have the latest version (currently 4.0.9.2), or you could try Shift-clicking Tools to show additional options.

Now Shift-click anywhere you could normally drag-and-drop a block, and start typing when the white horizontal bar starts flashing. As you type, the blocks displayed on the left will match what you've typed, so 'f o r e' will just leave the

“ RIGHT-CLICKING ANYWHERE IN A SCRIPT AREA BRINGS UP A MENU WITH THE MAGIC WORDS 'MAKE A BLOCK'

'forever' block, outlined in white. Press Enter; this block is inserted into your script space, and the bar moves inside the block. Now type 'm o v' to get the [move () steps]; press Enter and this time the highlight is in the number of steps to take. Press Enter, type the number of steps, and press Enter again. Now move the cursor down, type 't u' then Enter (or cursor down + Enter). Press Enter, type



■ Drag-and-drop code blocks in exactly the same way as normal in Snap!.

the number of degrees, and press Enter again. Finally, Ctrl-Shift-Enter will execute the current block.

You can press Tab or Shift-Tab to move around the blocks on the screen and then edit; the space bar shows you a menu of options, such as in the (item [/1/last/random] of []), and Escape stops editing.

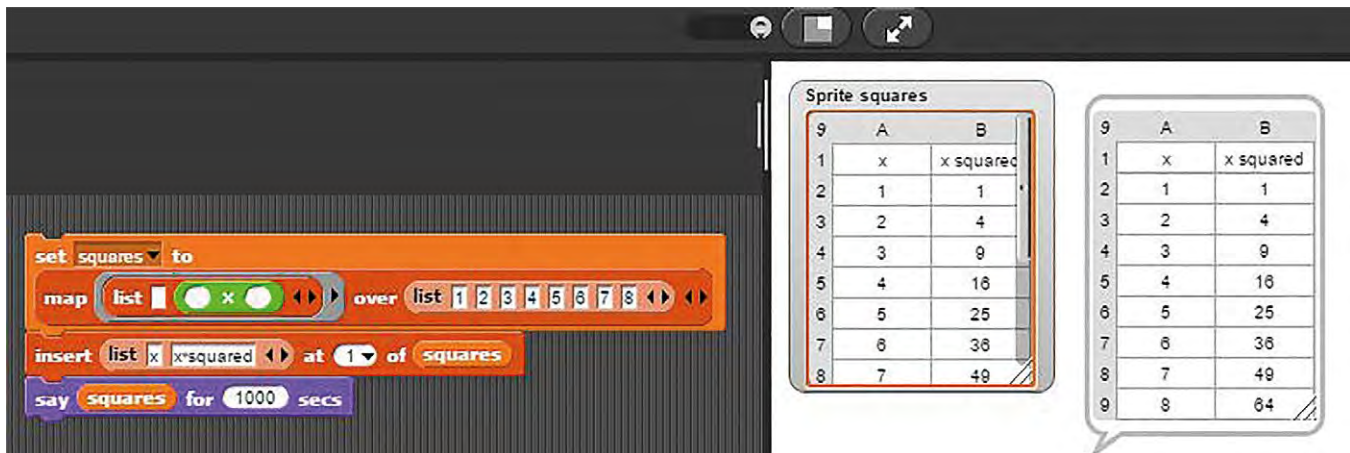
If you've enabled Table support, then creating a list that contains other lists will display as a table, rather than the conventional 'list of lists' format (see 'Table display in Snap!'). You can manipulate the column width and row height in this view by dragging the column/row labels; by manipulating the list data using the functional programming facilities introduced later in 'Scheme disguised as Scratch', you can swap columns, manipulate the values in columns, and so on. The documentation available on the Snap! website for tables shows you how to accomplish some of

the image manipulation (Photoshop-like filters) exercises, popularised by Mark Guzdial in his Media Computation course.

If you've enabled Visual stepping, then you may have noticed a small slider in the menu bar. Drag it all the way over to the left, and you have single stepping through your code. The middle control button is used to step to the next evaluation; incidentally, using this on the example code shows up an inefficiency! Move further to the right, and you trace through your code at a speed you control.

This has been a rapid introduction to Snap!. Subjects not covered include:

- Connecting to external devices e.g. Lego NXT or Arduino
- Saving your projects
- Connecting to the Snap! cloud (one missing feature is publicising your projects in a gallery, as Scratch does) ▶



- Systems based on Snap! like Edgy and Cellular for graphs and cellular automata, and Beetle Blocks/TurtleStitch for 3D printing and embroidery.

The Snap! website has a lot more documentation, and there are resources on the CAS site as well. Enjoy yourself!

is Scheme disguised as Scratch'.

When run, the code in the Snap! screen image repeatedly runs a random action from the list of four actions, and so the sprite does a drunkard's walk across the screen. Was it obvious? Conventionally, you'd do this by picking a random number, testing it in a series of [if then else] blocks, and executing one of the move/turn

are used to. The only thing we have to do is to 'ringify' the commands, which stops them being evaluated too early.

Snap! allows you to import a standard library. This includes 'map', to return a list that's had a function applied to every element of another list e.g. (map (square []) (list 1 2 3)) returns (list 1 4 9); 'filter', which returns a list of those elements of another list that satisfy a function, e.g. (select (even? []) (list 1 2 3 4 5 6 7 8)) returns (list 2 4 6 8). Furthermore, you can examine the code for these functions, as they're built using standard Snap! commands and functions.

Most of them don't have any loops, so goodbye, off-by-one errors! They're also shorter, and so more easily understood and verified than their conventional equivalents would be. (HW)

“ THE SNAP! WEBSITE HAS A LOT MORE DOCUMENTATION, AND THERE ARE RESOURCES ON THE CAS SITE AS WELL

Scheme disguised as Scratch

The website describes Snap! as combining:

Scratch's

- Drag-and-drop interface
- Visual metaphors for loops, conditionals, etc.
- Easy animation tools

with:

Scheme's

- First-class procedures
- First-class lists
- First-class objects (sprites)
- First-class continuations

More simply, as described by Brian Harvey (one of Snap!'s designers), 'Snap!

commands; imagine the changes you'd need to add a pen up/down command. By allowing procedures (either built-in or user-defined ones) to be used in the same way as other data, as first-class items, we can:

- Assign them to variables
- Store them in lists
- Pass them as parameters to procedures (see map and filter below) or
- Return them as values from functions

and we gain immeasurable power by doing so.

This is functional programming, but Snap! makes this (almost) as easy-to-use as any other programming technique by using all the visual metaphors students



■ Here we define the name of the new block, its category, and whether it's a command, function, or predicate.

a big year for

CAS Barefoot!

The Barefoot Computing Project helps support primary teachers with understanding and teaching computing.

This is a core part of our commitment to Tech Literacy, enabling young people to explore, discover and embrace an increasingly digital future.

Supported by



This is hugely important, as computing skills will increasingly be a prerequisite for everything young people do; from creative expression through to international commerce.

Since its launch over two years ago – Barefoot has gone from strength to strength, with the latest results including:



33,000 teachers supported

130,000+ downloads of resources



950,000 children reached



Teachers over **95%** more confident to teach 'computational thinking' after a workshop

To book your free workshop, download from our range of resources or become a volunteer, please visit

www.barefootcas.org.uk



10 TIPS

STARTING A MAKERSPACE

Creating a makerspace can be a pretty daunting task, but **Nick Provenzano**, author of *Your Starter Guide to Makerspaces*, is here with ten vital tips to get you started in the maker movement...

1 Talk to students

Student involvement is key to the success of this space. Ask students what a makerspace looks like to them. The students need to feel like they are invested in the process from the very beginning to take ownership of the makerspace. As an educator, you will discover what the students are passionate about, and will be able to tailor the makerspace to them.

2 Rally the staff

For a project like a makerspace, it's crucial to have the support of other teachers and administration. They can help you raise funds, deal with any administrative bumps in the road, and anything else that comes up. Makerspaces are about building a community of makers, and you want the entire staff to be part of this process.

4 Start designing

Now that you have a space, you will need to design it. Again, this is where student ownership is key. Have students draw their ideal space. Have them talk about colours, seating, charging stations, work areas, and so on. These designs will come in handy when it comes to funding, and the students will really take pride in using a space that they designed.



“FOR A PROJECT LIKE A MAKERSPACE, IT'S CRUCIAL TO HAVE THE SUPPORT OF OTHER TEACHERS AND ADMINISTRATION

3 Find a space

Finding space can be tough in schools. Makerspaces are great because they are so flexible; they will fit any space if you're creative enough. One word of caution: don't place the makerspace behind a locked door. A makerspace should be accessible to students all of the time.

5 Raise funds

Changing an entire space can be costly. Start by looking at government grants and donations from local businesses. Large local companies often love to support STEAM work at schools, and this is a great way for them to help. Fundraisers in the school are another great way to raise the money you need.



Parents love to support new projects that help their children.

6 Don't spend it all

People think they have to spend all of the money they get for a space right away. This is not true. You always want to save money to be used as needed for student projects. Students can pitch projects to staff and get funding for their own work in the space.

7 Let students have fun

Once the space is up and running, it's important not to over-schedule events or take up the space too much. You want a free-flowing space that allows students to come in and make when the mood strikes. Let the students try out the new equipment and take risks. They will have fun and will really embrace the makerspace as their own.

8 Review your curriculum

This one is not nearly as fun as the others, but it's very important. Once the makerspace is established, how will teachers adjust their instruction to allow

for a more inquiry-based curriculum? This allows for students to use the makerspace to create projects and bring them to class. The ultimate goal of a makerspace is to become a seamless part of the school. Adjusting the curriculum to encourage more creation is a way to do that.

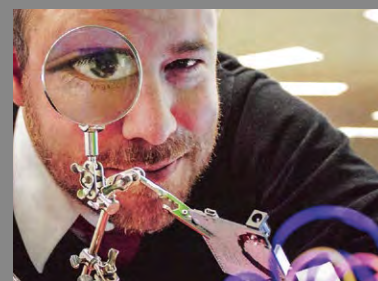
9 Be OK with failure

It's very important that everyone involved is ready to embrace failures. Makerspaces need to be a special place where failures can be celebrated as learning moments. Let students and staff fail in the space as they try new things, and let everyone know that failure is proof that they are learning.

10 Make connections

One of the many things that make makerspaces so great is the maker community. There are makers all over the world and it's easy to connect with them. If you are on Twitter, follow #MakerEd to see the great things others are doing and reach out with any questions. The collective group of makers online have a wealth of knowledge. Tap into it and, eventually, you will be the one offering advice to others who need help.

These steps do not encompass everything it takes to get a makerspace up and running smoothly, but they are a great start for those looking to get involved in the world of making. You can find more detailed info and resources in Your Starter Guide to Makerspaces. Good luck and happy making! (HW)



Nicholas Provenzano is a high-school English teacher, author, speaker, and consultant. He has been featured on CNN, com, Education Week, The New York Times, and other media outlets. In 2013, he was awarded the Technology Teacher of the Year by MACUL and ISTE. Nicholas is also a Google Certified Innovator, Raspberry Pi Certified Educator, and a TEEd Innovative Educator. His new book, *Your Starter Guide to Makerspaces* (helloworld.cc/2iyoYUx), was a best-seller and can be found on **Amazon.com**. Find him on Twitter at **@TheNerdyTeacher**.

BLUFFER'S GUIDE TO 3D PRINTERS

3D printing is revolutionising art, science, and medicine, but can it do the same for teaching computing and digital making? CoderDojo mentor Richard Hayler investigates...

What is 3D printing?



3D printing is a manufacturing process which produces solid objects from digital 3D models. It's called "additive manufacturing" in industry. There are several types of 3D printing techniques, but the most common method is to extrude molten filament

(a plastic-like substance) to build up a model in layers.

What can students make with a 3D printer?

Students can create 3D-printed objects using design software, or they can download templates and use them as part of their studies.

A good example is the GB3D Type Fossils (helloworld.cc/2iyazN3). This free collection of templates contains 1,800 fossils from British museums.

History students have also printed working models of trebuchets, and design students develop their own versions of items they might own (such as phone cases or tablet stands).

How big can 3D-printed objects be?

For practical, time-saving and economic, reasons 3D printed objects tend to be less than 25cm square.

What software will I need?

Students can use 3D CAD (Computer-Aided Design) software to examine and design models for 3D printing.

Many schools already use SketchUp (helloworld.cc/2jVN6RV) to make 3D designs. Blender (helloworld.cc/2iLclZA) is another solid option.

SketchUp Make is free and comes with handy templates to help ensure that designs are the right scale.

How do students design models to be 3D printed?

The models that students design are exported as STL (STereoLithography) files. These are the files that students will also download online.

The exported STL file is loaded into another program that works out how to divide the object into layers, a process known as slicing.

Cura, a free program from Ultimaker, is one of the most popular (helloworld.cc/2jkXJ1K).

The output of this process is a G-Code format file which is read by the printer. Some 3D printers accept files directly; others require you to export the file to an SD card, which you then insert into the 3D printer.

How long does it take to learn 3D printing?

According to the DfE: "Many of the project teachers reported that it took a few months to become familiar enough with the printer and associated software to use it successfully and confidently in teaching. Integrating use of the 3D printer into the curriculum proved most successful with self-confident teachers who were passionate about their subject, and not afraid to experiment and innovate."

Are there free training resources?



There are some good online training resources for learning 3D printing. "How To Use A 3D Printer" by ALISON (helloworld.cc/2k3sXh7) is a respected free course. Lynda offers

a more detailed course, called "Up and Running with 3D Printing" (helloworld.cc/2jJgf3D). Lynda charges a monthly subscription, but you get 10 days free, plenty of time to view just the one course.

3D PRINTERS AND THE CURRICULUM

Can I use the 3D printer across the curriculum?



Just like all digital skills, 3D printing should be an option for any 'making' project: design and print a model of the Grand

Canyon for geography, create various prisms for mathematics, recreate a castle's turret for history, or produce aerodynamic stomp rocket fairings in science. And, of course, every design you provide for the printer can be an exploration of engineering and materials science.

Do 3D printers help teach STEM?

3D printers entered schools in the Design and Technology (DT) departments, so there's a tendency to think of it in design terms. However, a DfE report in 2013 (helloworld.cc/2iDYbX2) found "considerable potential" for use in a range of STEM subjects. You can use 3D printers as a "link between mathematics, design, and physics."

How much do 3D printers cost?



Prices vary, depending on what type and size you buy, but it's a large outlay. Schools can purchase a quality printer that can produce

15cm objects for about £800. Larger, top-end machines cost £1,000 or £2,000. Running costs need to be taken into account.

What are the running costs?

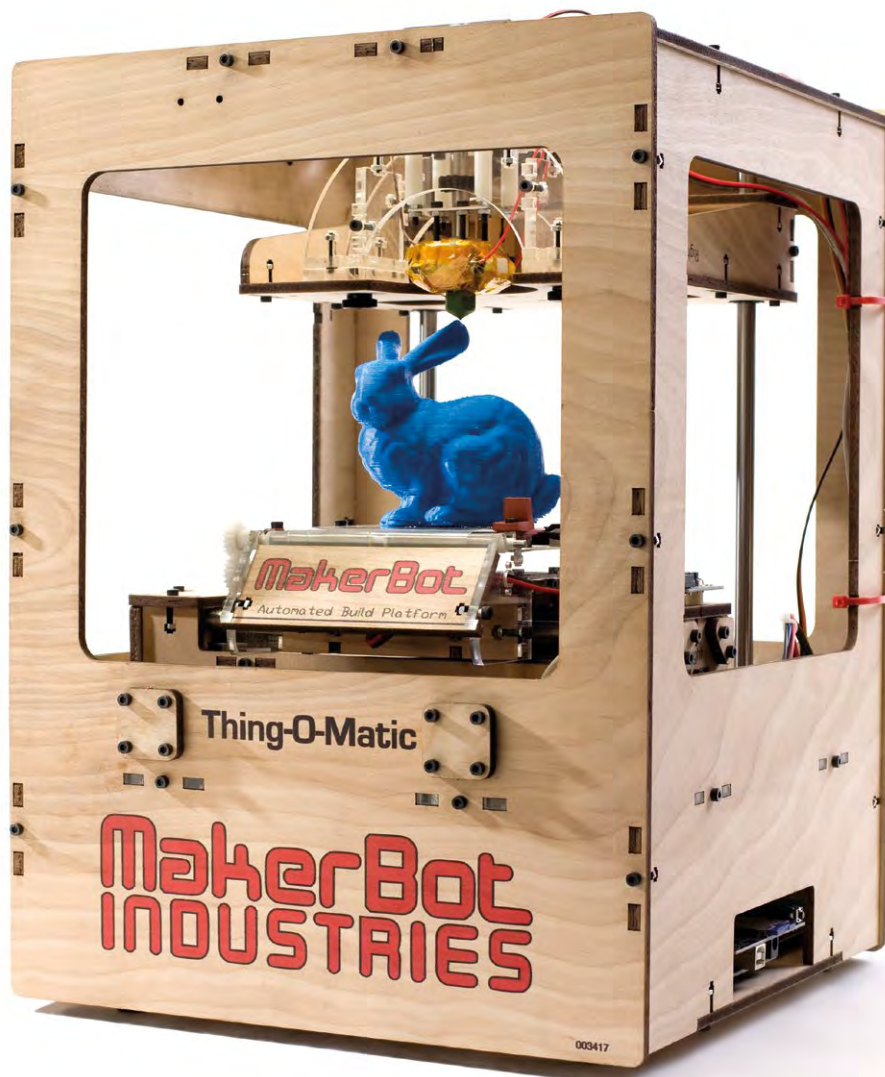
The main cost is the filament. Typical costs are around £25 for 1kg of filament. This amount makes, unsurprisingly, 1kg of 3D-printed objects. A MakerBot test produced 392 chess pieces from 1kg of filament (helloworld.cc/2iyhg1q). How much filament is used for a given print job will depend on the size and amount of solid material used in the design. Most software packages offer some idea of how much filament you are using.

Can I move a printer between classes?

3D printers are generally light enough to be picked up and moved around. But they can be fragile, so they will need to be handled with care. In general, we'd advise keeping it in a single, secure location.

How secure are they?

3D printers are small enough to pick up, and expensive enough to go wandering. Theft is a concern; be sure to keep it under lock and key.



■ Printed objects are sculpted one layer at a time. The process is slow, but hugely rewarding for students

What do students think?

It's a hugely exciting area. Children love to explore and create, and 3D printers enable them to turn ideas into creations.

The process of 3D printing lends itself to the concept of iteration (design, prototype, evaluate, and redesign). It encourages higher-level thinking, and a more realistic experience of design and manufacture in the real world.

Is it an employable skill?

Companies have been using 3D printing since the 1980s and usage is on the increase. PricewaterhouseCoopers surveyed 100 leading manufacturing companies and found two-thirds already using 3D printing (helloworld.cc/2iyiD0j). So it's a handy, if still quite rare, skill for students to acquire. ■



■ 3D printers have hot nozzles and moving parts, and are noisy. In general, it's best to get one that keeps its moving components inside a box

■ Any health and safety issues?

3D printers are safe. However, as with anything that has moving parts, there's always a risk of people getting fingers trapped. 3D printers move quickly, so hands should be kept out of the print zone when the device is operating.

Also, the print head and bed do get hot, typically 210 degrees and 60 degrees respectively for ABS and PLA filament. While they aren't a fire risk, we wouldn't advise putting paper or other flammable materials nearby.

If the printer is going to be used by children, then a machine like the CEL Robox might be a good idea, as it has a completely enclosed print zone and a door that locks when operating.

Most filament materials don't give off any harmful fumes, but it's best to use them in an open, well-ventilated area, especially if you are using the 3D printer for extended periods.

Can students use one unsupervised?

We would advise you to only allow supervised access to 3D printing, or to take direct control of the printing. 3D printers run very hot, have moving parts, and are fragile and easily broken. Caution is recommended.

“ There's no getting away from it: 3D printers are noisy, especially in models where the nozzle is exposed.

How noisy are they?

There's no getting away from it: 3D printers are noisy, especially models where the print nozzle is exposed. Even a single device can be distracting. You'll have to shout over multiple 3D printers being used at once.

Some of the enclosed printers are quiet enough to be left working away at the back of a classroom, but the noise from those without side panels will be distracting. Make sure you take noise levels into account when purchasing a model, and get one with a sealed unit.

How long does a 3D print job take?

The slow speed of 3D printers is a commonly reported frustration among users. You can print a small trinket item in 20 minutes on a fast, low-resolution setting. But it's wiser to plan at least an hour for each 3D-printed object.

With a class of students, it's wise to try to manage their (and your own) expectations. Consider forming study groups with three or four children.



■ This string-like substance is known as “filament” and is the material used to 3D print objects. Large objects use up a lot of filament, so it's a good idea to keep objects as small as possible

What are the most common problems?

Things get stuck and jammed frequently. Typically, this is the filament, which can clog up the print head nozzle. Most machines will come with tools for removing lodged filament. It helps if teachers have a 'maker approach' and are prepared to unclog nozzles without calling in for a service team.

What kind of filament should I use?



The most popular filaments used for 3D printing are ABS (Acrylonitrile Butadiene Styrene) and PLA (Polylactic Acid).

ABS is prone to shrinkage as it cools, leading to warping of the lower parts of the printed object.

PLA is plant-based and, therefore, biodegradable. It's also less smelly, but is stickier and leads to more clogged nozzle problems.

Where can I find objects to 3D print?

There are some websites where makers can upload and share their designs. The most well-known and largest are Thingiverse (helloworld.cc/2k1QS0n) and Yeggi (helloworld.cc/2jkQWoB).

How should I choose the best one for my school?

For an unbiased review of printers, this 3D Hubs best printer guide is a good place to start (helloworld.cc/2jGSXvb).

Look for a "makerspace" in your local area; these are clubs springing up around the country, that offer communal access to equipment like 3D printers.



■ Students use 3D printers to iteratively build and test components, like these working fans

The people at a makerspace will have experience with using 3D printers, and be able to offer you advice on buying and using a 3D printer for school. They can also offer you a hands-on experience of 3D printing and potentially some training.

Is 3D printing worth it?

3D printing technology is becoming more mature all the time, but there remains an element of fiddliness to the printers. Unreliability is one reason they haven't become more widely used by the general public.

You will almost certainly have to troubleshoot problems and unexpected results from time to time. The end results are hugely rewarding, though, and it's one of the most engaging aspects of modern technology. If you take the 3D printing leap of faith, you won't regret it; your students will thank you. (HW)

■ Students can use 3D printers to design and build real-world objects like this vase

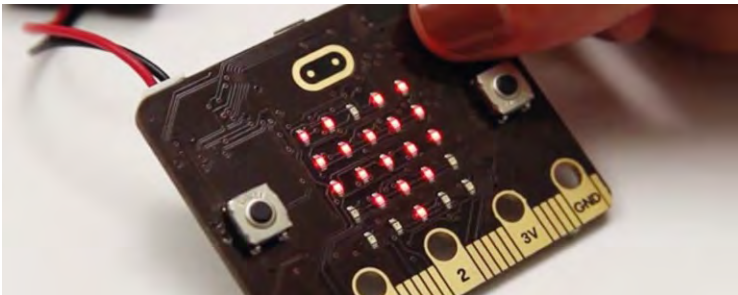


GOT A QUESTION?

This FAQ is made up of genuine teacher questions. If you've got a question you can ask us on Twitter via @HelloWorld_Edu or using the #HelloWorld hashtag. Alternatively, email us (contact@helloworld.cc) with 'Teacher Question' in the subject line.

YOUR QUESTIONS ANSWERED

Teaching computing and digital making requires a whole new set of skills, and understanding the various options and requirements can be incredibly challenging. Don't be afraid to put your hand up and ask us anything...



Q What's the best way to start teaching physical computing?

A Controlling or simulating physical systems is a requirement in Key Stage 2. Physical computing is concerned with real-world hardware, rather than just manipulating pixels on a screen.

It's about teaching kids the "physical mechanics and real-life applications of devices" rather than just the virtual world of software. It's hands-on, so more fun for students, but it's also practical.

It's also a lot tougher for teachers to teach than software. There's the expense of buying the physical kit, and you need time and resources to learn how to use the equipment yourself (before teaching the blighters how to use it).

It's a challenge, but there is a lot of help out there. Small, inexpensive

devices like the BBC Micro:bit and Raspberry Pi are inexpensive pieces of hardware designed to make it easier to get physical computing into the classroom. There are also lots of teaching resources for both devices.

Start by using a simple program like Scratch to flash the LED lights on a Micro:bit board (follow this teaching resource: helloworld.cc/2iAUS7l). Or set up the operating system on a Raspberry Pi (helloworld.cc/2jyr8HN) board.

After that, you keep learning from the teacher resources on how to build simple electronic circuits. You can make traffic lights and buzzer circuits. They allow pupils to engage with wearable technology, robotics, and other forms of digital making.



Q I'VE DISCOVERED A BUNCH OF OLD RASPBERRY PI MODEL B BOARDS. SHOULD I THROW THEM OUT?

A Oh no! You can still do loads with these trusty Pi devices. You can still run the latest version of the operating system and connect to external hardware using the pins on the board.

They run a little slower than the newer models, but they're still great for learning physical computing.

They are particularly good for use in electronics projects (see the "what's the best way to introduce robotics" question on the next page).

The only drawback is that you can't connect the latest HAT hardware equipment to these older boards (they don't have enough pins). But that would be an extra expense anyway.

Your students can use them to learn programming and physical computing. You can attach electronic components to the pins on the board. Take a look at the Raspberry Pi Teacher's Guide (helloworld.cc/2jZIIYE).

You could even use the old boards to create wearable projects.

Q What, exactly, is computational thinking and why is it so important for students to understand?

A It's a significant term at the moment, and you're totally right to ask what it's all about. Indeed, "computational thinking" is at the start of the National Curriculum, and it weaves in and out the whole programme.

"A high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world," says the DfE.

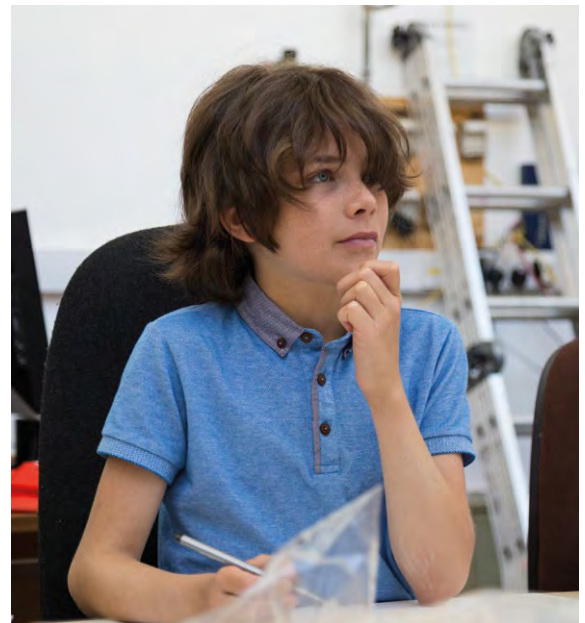
Like many high-end concepts, Computational Thinking is surprisingly hard to pin down when you talk about it. Jeannette Wing's original definition was: "Computational

thinking is the thought processes involved in formulating problems and their solutions."

Computational Thinking is something that humans do; not machines.

Broadly speaking, it's a collection of skills that enable students to interact with computers. It includes the ability to think logically and understand algorithms. At higher levels, it includes concepts like abstraction and recursion.

The UK Forum for Computing Education has this PDF document. The PDF explains Computational Thinking in more detail (helloworld.cc/2ilhFcW).



Computational thinking
Teach children the thought processes they need to interact with computers

Q Moving on from Scratch to a text-based programming language is a big jump. What is the best way of easing the transition for children?

A A Key Stage 3 requirement is to "use 2 or more programming languages, at least one of which is textual." This condition almost certainly means adding a second language on top of Scratch.

Typically this second language is Python. Compared to other languages, Python is easy to understand and friendly enough to introduce to 11-year-olds. It's not that difficult; students just find it a bit dull. Having spent a lot of time dragging highly visual blocks around your students will find the dusty world of text-based programming a huge letdown.

There are lots of approaches to make Python more exciting. One is to use an interim language, like Google's Blockly (helloworld.cc/2jwBPuZ). This program is like Scratch, but it converts visual blocks into Python



code. It may just be delaying the inevitable, though.

Another option is to move into physical computing quickly. Either introduce a Micro:bit or a Raspberry Pi into the classroom. This hardware replaces the visual stimulus of Scratch characters and kids see Python doing stuff.

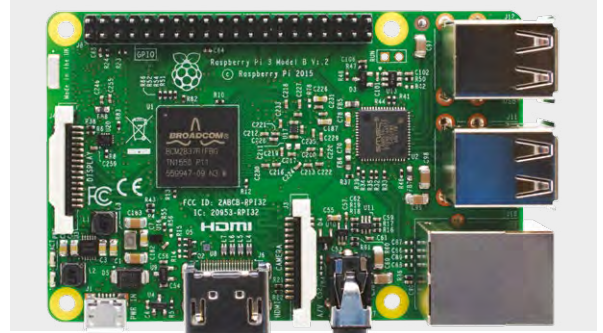
You can also use Python inside Minecraft on a Raspberry Pi. This approach lets children use Python in a 3D visual world. See this "Getting Started with Minecraft Pi" resource (helloworld.cc/2ilkKtr).

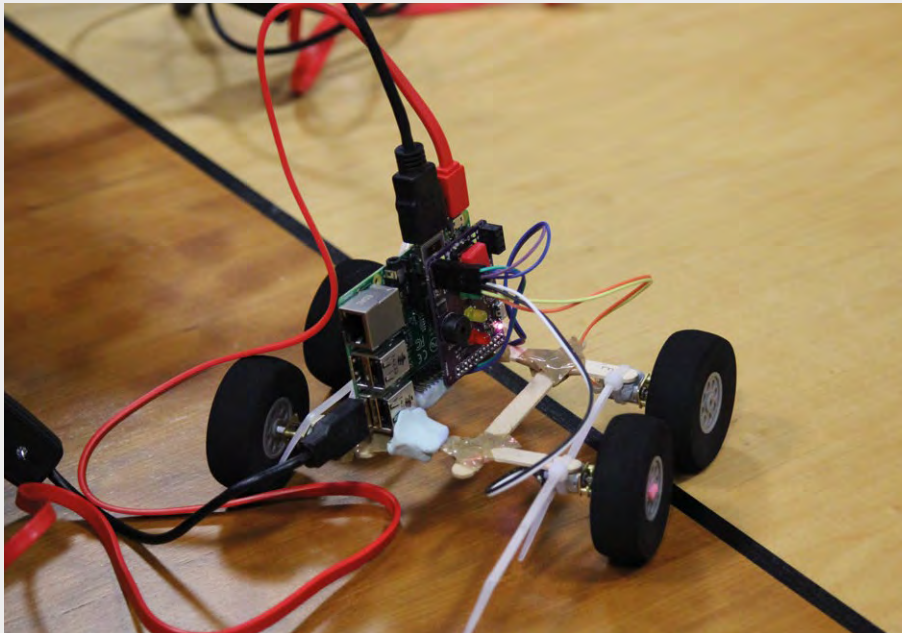
Q HOW IS A MICRO:BIT DIFFERENT FROM A RASPBERRY PI?

A In many respects, both are small devices trying to solve the same problem. They both make it affordable and easy to introduce children to physical computing.

They work very differently, though. With a Micro:bit, students create programs using a web interface on a regular computer. These are then transferred to the Micro:bit using a USB cable.

The Raspberry Pi is a complete computer on a single small board. It has an SD Card for storing files on and can be connected to a display. You can plug in a keyboard and mouse to a Raspberry Pi, and students use an operating system similar to that on Mac or Windows computers.





Q WHAT'S THE BEST WAY TO INTRODUCE ROBOTICS TO MY CLASS?

A Robotics is a fantastic project to present: if you can handle the complexity of the equipment. Robots are exciting to build and highly entertaining.

Robots use two types of motors: DC Motors and Servos. You'll normally need a Raspberry Pi and a separate piece of hardware, known as a Driver Board to control the motors

DC (direct current) motors move robots around. When you supply power, the motor will start turning until it's switched off. Reversing the polarity of the supplied voltage will reverse the direction. You attach two motors to a chassis (this can be anything box

shaped) and use Python or Scratch to turn the motors on and off. This makes the robot move forwards, backwards, left and right.

Servo motors are controlled to move to a precise angle and normally have three wires. Two wires supply power, while the third is the control line. These are used to build robotic arms.

There are lots of robotics kits around, and it's certainly worth buying a kit rather than starting from scratch. A cost-effective way to start is using a CamJam EduKit #3 kit (£18, helloworld.cc/2jZGzG1). Maplin's sells a Robotic Arm Kit for £29 (helloworld.cc/2iQwqxXD).

Q Should we be teaching our pupils Python 3 or Python 2?

A There's not much between the two versions of Python. Unfortunately, the one command students use the most: "print" works differently in each version. So most programs created in Python 2 are not compatible with Python 3, and vice versa.

The good news is that apart from that, they're very similar. So no matter which version of Python you choose. Your students aren't going to end up with a vastly different understanding of Python (or coding in general).

A few years ago we'd have suggested you stick with Python 2, but these days more and more resources are written using Python 3. Micro:bit uses a full reimplement of Python 3, and The Raspberry Pi Foundation produces teaching resources in Python 3. So while you may find a lot of old code around in Python 2, we think you should start with Python 3.

Python charmer

Python is the easiest programming language to learn after Scratch

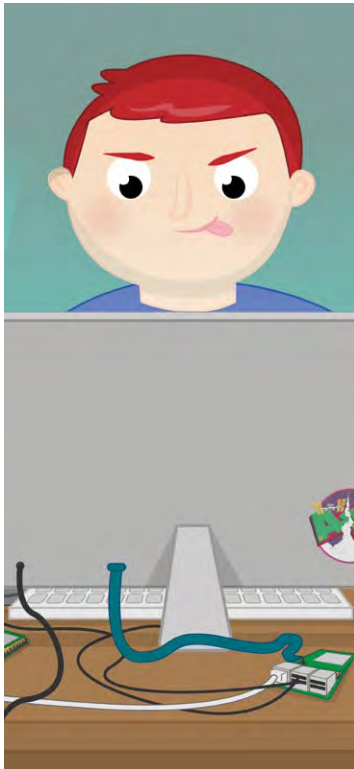


Q Should I teach HTML and CSS before moving to Python?

A HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are the two languages used to create websites. They're both text-based languages, and there's a real temptation to use them instead of a language like Python to meet the "text-based"

remit of the national curriculum.

Don't be fooled into thinking that HTML or CSS is an adequate substitute for Python. You can't teach computational thinking using HTML. It's better to move straight from a visual environment like Scratch direct to Python.



Q The UK National Curriculum for Computing refers to ‘sequence, selection, and repetition’. What do these terms mean?

A These are the three mainstays of computer programming. They’re also known as procedure, branching and looping.

The sequence is all about doing things in the correct order. If you try to pour water from a bottle before removing the lid, nothing will come out. Programs run one line of code at a time. And some lines of code require others to come first (you can’t increase the amount of variable if you haven’t already created it, for example).

Start at the beginning

All programs are built up of simple steps of sequence, selection and repetition. Once you know these basics you can create all kinds of programs

Selection is when a program makes decisions based on available information (or data). A program could know the weight of dogs and say “yip” for small dogs, or “woof” for big ones. It selects based on the data.

Repetition is, as the name implies, about repeating actions. A program could count from 1 to 100. But instead of writing one-hundred print statements you’d write one, and have the program run it one-hundred times (increasing the amount by one each time).



Q How am I supposed to teach algorithms to five-year-olds?

A Okay, don’t panic. We blame Hollywood for a lot of the mystique surrounding the word “algorithm” (it brings to mind Iron Man creating AI suits or Mark Zuckerberg creating super-secret formulas).

The reality is a lot less glamorous. It’s just doing something one step at a time.

An algorithm is just the sequence of precise and unambiguous steps

a computer takes to do something. You can introduce this idea to young children using activity sheets. Typical activities include finding the way home on a map (turn left, go forward, turn right, and so on). Or the steps you’d take to make a cup of tea (put water in the kettle, boil the kettle, put a tea bag in the cup, and so on). Twinkl has some worksheets you can use (helloworld.cc/2jwHOjg).

Q WHAT IS A GOOD FIRST CLASS PROJECT FOR 3D PRINTING?

A Start with something small and straightforward, like a trinket or badge. Give the pupils a template file in SketchUp that contains the basic design at the correct dimensions (a 25mm diameter circular plate, for example), and have them customise it by adding 3D shapes or text. With an object that size, you should be able to print at least 6 in a single run, so that the pupils can see the fruits of their labours reasonably quickly.



PROGRAMMING APPS FOR IPAD

Using iPads in class? Get a boost with these recommended coding apps

Technology is always marching forward, and one of the greatest innovations in recent times is the portable tablet computer. The iPad in particular is incredibly user-friendly and loved the world over, especially by

young people. It makes complete sense, then, that as the world of technology changes, the world of computing education changes with it. That brings us to this list of fantastic apps that can help teach coding on an iPad. **(HW)**



QUICK FACTS

- Apple's own coding app
- Learn coding through puzzles
- Optimised for touch

SWIFT PLAYGROUNDS

IN FO FROM Apple | PRICE FREE | URL magpi.cc/2j155Y1

Swift Playgrounds is Apple's own attempt at providing coding education through an app. It does this by offering coding puzzles for you to solve that can be immediately run on the little 3D cube world living in the app.

Swift Playgrounds concentrates more on writing code than moving

around blocks of code, distinguishing it from Tynker on this app list. It also makes the app feel like an interactive book of coding, which is a cool aesthetic. Once a student has completed several puzzles and challenges, they can have a stab at actually making some code themselves without any prompts. **(HW)**



QUICK FACTS

- Program with code blocks
- Comes with lots of tutorials
- Easy to make games with

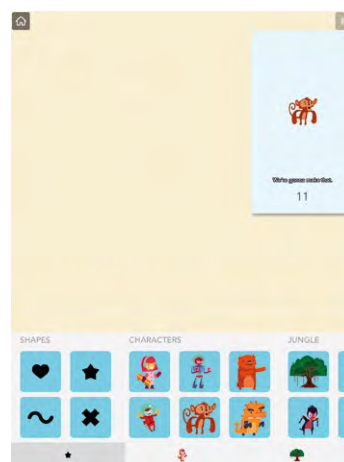
HOPSCOTCH

IN FO FROM Apple | PRICE FREE | URL magpi.cc/2j155Y1

Hopscotch is possibly one of the most basic and simple apps on this list, but that doesn't mean it's too simple to use. Much like Scratch, it uses blocks of code to create programs and games in an attempt to teach programming logic to students.

There are lots of games and

tutorials that come with the app, and there's also a monthly subscription you can get (£6/\$8), which adds more tutorials and ideas on a monthly basis. This includes examples of currently popular games, like Pokemon GO, that should keep students interested much longer than other similar apps. **(HW)**





CODEA

IN FO

FROM Two Lives Left | PRICE £11 / \$15 | URL magpi.cc/2j16ga0

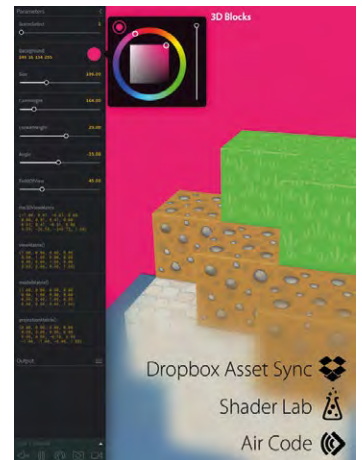
QUICK FACTS

- Codea lets you program in Lua
- You can run programs like apps
- Edit code with touchscreen gestures

This app is a little more advanced but still has some of the more interesting, user-friendly features of the others. It still lets you make programs and games, but this time it lets you use the Lua programming language to create apps. This code can then be manipulated through touch (such

as changing colour or the way a sound works), and then exported as an actual app for your device when you're happy with it.

There's plenty of other neat additions (remote coding, video recording of how your code works), and there's some example projects to learn from as well. **(HW)**



CODE2GO

IN FO

FROM Nathaniel Herman
PRICE £2 / \$3
URL magpi.cc/2icWPao

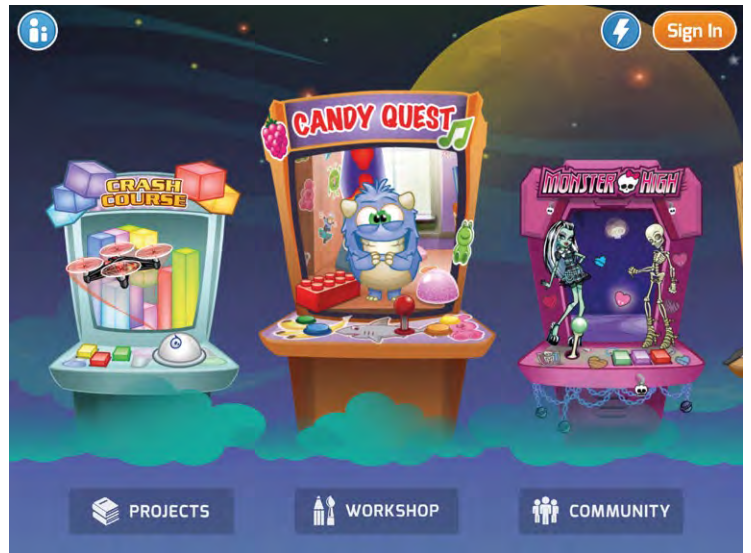
QUICK FACTS

- Write in any type of programming language
- Runs code so you can see if it works
- Can also render web pages

Where to go once you're done with simple block coding and coding-lite apps like the others in this list? Code2Go is a full text editor that lets you program in all major languages, including highlighting for the

code that looks the same as official development environments. Students who write code on here can then test it by running it using the online service provided, which allows for checking and debugging of the code.

It also has a function that lets you render web pages, making it a great way for students to play around with HTML and website-building. **(HW)**



TYNKER

IN FO

FROM Tynker | PRICE FREE | URL magpi.cc/2icWsfC

Tynker is basically Scratch for iPad (although you can access Scratch via the browser these days); however, it does come with a lot of extra stuff to make it worthwhile. As well as being able to use blocks of code to create programs and games, there are built-in games to play that require students to perform coding tasks to proceed, as well as over 100 step-by-step tutorials to help further knowledge in the basics of programming.

Apps and games made on

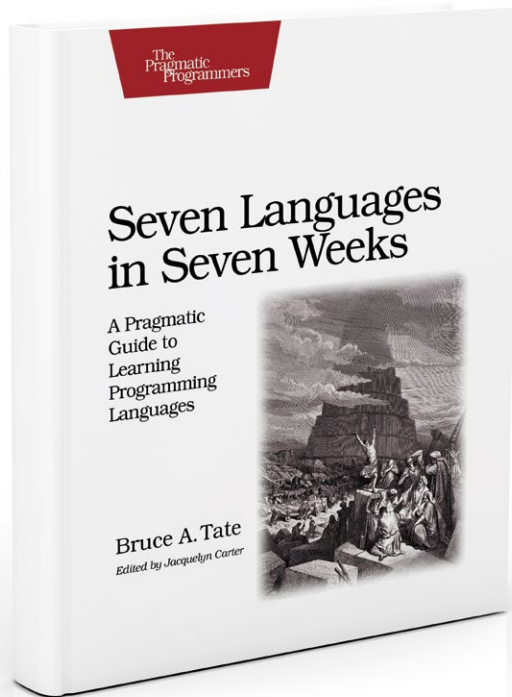
QUICK FACTS

- Works like Scratch
- Teaches coding through games
- Works with some smartphone-connected toys



Tynker can also be shared online, making it a great way to get students excited by being part of a bigger coding community. **(HW)**





SEVEN LANGUAGES IN SEVEN WEEKS

John Stout reviews a book to take you beyond 'Hello, World!' in seven different programming languages, considering each one's unique strengths and weaknesses

INFO

BY Bruce A Tate | PUBLISHER Pragmatic Programmers, LLC. | PRICE £21.74 | ISBN 1-934356-59-X | URL helloworld.cc/2jnobYI

Bruce Tate takes you through the basics of seven different programming languages (Ruby, Io, Prolog, Scala, Erlang, Clojure, and Haskell) and shows you what makes each one special. In the foreword we're told that 'Learning to program is like learning to swim. No amount of theory is a substitute for diving into the pool ...' and that's really what this book does: dive in!

much less than 24 hours) spent on an introduction to the language, to which paradigm(s) it belongs, and its history. Day 2 is spent introducing yourself to the language: how you enter expressions, strings, Booleans, and so on, and how it handles types (strong vs. weak, static vs. dynamic).

Day 3 takes you deeper into each language, and sometimes the pace takes your breath away, but there's

Interviews with designers

One of the most useful parts of each chapter are the interviews with either a designer of the language (Haskell has two, with Philip Wadler and the CAS chair, Simon Peyton Jones), or an expert user. You get to know why the language was developed (why do we develop new languages? Don't we have enough?), what they consider important, and often what they regret and would like to change.

Day 4 is where each language really comes alive, with substantial programs showing off the capabilities of the language.

Finally, you get a summary of what you've learned, and the author's views on that language's strengths and weaknesses.

You won't use this book as a tutorial for a single language, but as an introduction to programming languages it's superb; it would also make an interesting 7-week extension course for students. **(HW)**

“ Learning to program is like learning to swim. No amount of theory is a substitute for diving into the pool... **”**

At times you may find that you're drowning (for example, there's no help with installing any of the languages), but if you persevere you'll end up with a broad understanding, if not a deep one of any particular language.

Each language is dealt with in same way, with Day 1 (you'll need

lots of self-study questions and support on the Internet.

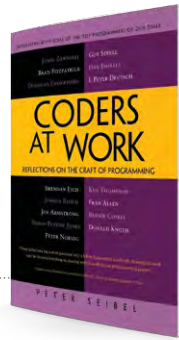
As you progress, you start finding concepts repeated, which really helps your understanding; if you haven't used things like recursion or pattern matching much, then Day 3 gives plenty of examples across the languages.

CODERS AT WORK

John Stout on Peter Seibel's interviews with 15 professional coders, discussing how they got into coding, what they like, and what they hate!

IN FO

BY Peter Seibel | PUBLISHER Apress | PRICE £15.32
ISBN 978-1-4302-1948-4 | URL codersatwork.com



In this book the emphasis is on the coder, not languages, so they each get a chapter to themselves. You'll learn how they got into coding, what they enjoy about it, and what keeps them coding (if they're still coding). You'll get a feeling for the person behind the language, program, or system, and some gossip about the other personalities and languages in the field; C++ gets a bit of stick here!

It's particularly nice to see two of the developers of Smalltalk/Squeak, the language in which Scratch was originally implemented.

In each chapter you're likely to find something that one of these coders feels about coding and think "thank heavens it's not just me!". My favourite is from chapter 8, the interview with Simon Peyton Jones, when he talks about being faced with some code that 'you wrote yourself but no longer dare to modify.' (HW)

BCS GLOSSARY

Julie-Anne Maisey reviews the BCS glossary, now in its 14th edition: the most comprehensive and authoritative reference of the vocabulary of computing.

IN FO

BY BCS Academy Glossary Working Party | PUBLISHER BCS, The Chartered Institute for IT
PRICE £19.99 | ISBN 9781780173269 | URL bcs.org/books/glossary



The BCS Glossary includes usage guidance and advice, which acknowledges the sometimes difficult aspects that overwhelm computing students. The glossary deals with jargon in a way that scaffolds and builds confidence in the reader: it begins using more general terms, before moving on to more specialised and detailed computing references, with examples and bold print to support key words. The glossary is useful at A level and beyond, but is accessible to GCSE students, as a guide to support learning, enhance

revision, and embed understanding of computing terms.

Figures, tables, and diagrams support and enhance the text, increasing engagement. These are also pitched at an appropriate level for all ages.

If computing is taught with a focus on exams, students would benefit from using this during theory lessons and as a revision tool.

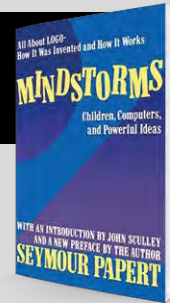
It's an important addition to any academic book collection, that will no doubt have bookmarks inserted and corners folded over the years to come. (HW)

ESSENTIAL READING:

If you've been inspired by our cover feature on Seymour Papert's legacy, here's our guide to the three books that best capture his vision for education

MINDSTORMS: CHILDREN, COMPUTERS, AND POWERFUL IDEAS

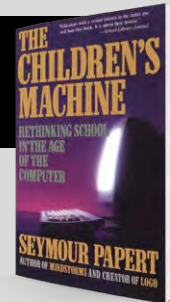
BY Seymour Papert
PUBLISHER Basic Books
PRICE £11.89
ISBN 978-0465046744
URL helloworld.cc/2jnDi4l



The 1980 classic on the first years of computer programming in education. Many of Papert's ideas here, such as thinking, literally walking through code, learning through making, and how debugging builds resilience are as valid today as they were revolutionary then.

THE CHILDREN'S MACHINE: RETHINKING SCHOOL IN THE AGE OF THE COMPUTER

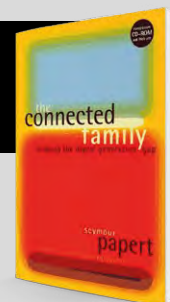
BY Seymour Papert
PUBLISHER Prentice Hall
PRICE £14.98
ISBN 978-0745016030
URL helloworld.cc/2jnBzMh



A visionary text, in which Papert looks at how schooling would change when children had access to the world's knowledge via computers of their own. Sets an agenda for game-based learning beyond drill and practice and rich, immersive multimedia.

THE CONNECTED FAMILY: BRIDGING THE DIGITAL GENERATION GAP

BY Sandra L Emerson
PUBLISHER Addison Wesley
PRICE £43.99
ISBN 978-0201703092
URL helloworld.cc/2jnAFiG



Papert recognised that the ubiquity of digital technology would move the place of learning from school to home, and that learning at home would be more about exploration and discovery than instruction and exercises. A helpful response to questions of how grandparents, parents, and children can learn and use technology together.



GREG MICHAELSON PROFESSOR OF COMPUTER SCIENCE

PEANO PLAYER

Breaking down the concept of programming languages, using simple maths...

Sometimes, programming languages are characterised as being compiled, interpreted, or interactive, but I think that this is wholly misleading. To explain why, I'm going to explore the implementation of a minimal programming language based on the early 20th-century mathematician Giuseppe Peano's simple but powerful model of arithmetic.

In Peano's system, integers are represented as finite successors of zero, and operations are defined with base cases for zero and inductive cases for non-zero numbers, much like how we write recursion. We might view Peano integers themselves as constituting a very simple programming language. We could define a number as a sequence of SUCCs ending with ZERO:

`number -> ZERO | SUCC number`

Thus, the first few numbers are ZERO, SUCC ZERO, SUCC SUCC ZERO, and so on.

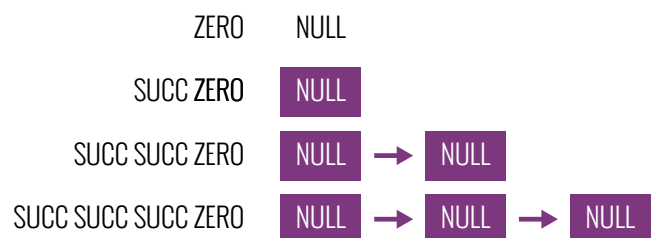
Suppose we want to run programs in this language, to output the equivalent decimal number. We'll start with the first two stages of any language implementation: lexical and syntactic analysis.

Our lexical analyser will recognise valid letter sequences, and output values to represent the corresponding symbol, say '0' for ZERO and '+' for SUCC. For example:

`S U C C S U C C S U C C Z E R O → +++0`

Next, our syntax analyser will input sequences of symbols and parse them; that is, check that they correspond to the grammar. As we parse the sequence, we'll build an internal representation, also known as an abstract syntax tree (AST).

Here we might use a very simple linked list for our AST, with one node for each SUCC. An empty list will represent ZERO. For example:



Our parser is:

```
RECORD number IS { number next }
DECLARE tree AS number IS NULL
DECLARE symbol IS <first in symbol sequence>
WHILE symbol != '0' DO
    SET tree TO number(tree)
    SET symbol TO <next in symbol sequence>
END WHILE
```


Finally, to implement our language, we could write an interpreter that starts with the value 0 then iterates through the AST, adding one for every SUCC node. At the end of the AST, it displays the final value:

```
DECLARE value IS 0
WHILE tree != NULL DO
  SET value TO value + 1
  SET tree TO tree.next
END WHILE
SEND value TO DISPLAY
```

So: `+++0` → 3

Of course, our interpreter is just finding the length of the list representing the AST.

Alternatively, we could write a compiler to generate target code that computes the decimal number from an expression. The compiler always starts by producing code to print a 0, and then iterates through the AST, generating an additional “+1” in the printed expression for every node. Let’s generate BASIC, my favourite language:

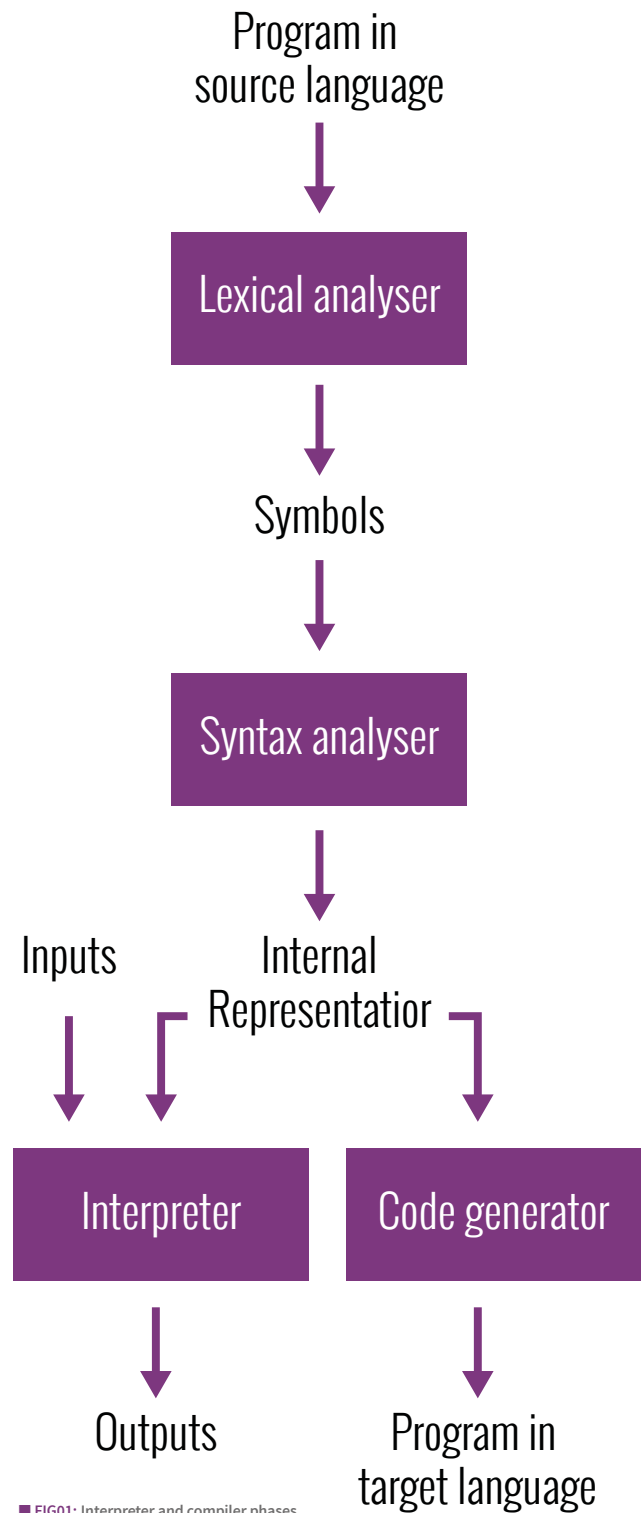
```
SEND "5 PRINT 0" TO DISPLAY
WHILE tree != NULL DO
  SEND "+1" TO DISPLAY
  SET tree TO tree.next
END WHILE
SEND "\n" TO DISPLAY
```

So: `+++0` → 5 PRINT 0+1+1+1

Notice that our interpreter and compiler are very similar in structure: both iterate through the abstract syntax tree, but where the interpreter does arithmetic directly, the compiler outputs new code that itself has to be compiled or interpreted to do the equivalent arithmetic.

We can use this minimal example of language processing to tease out a number of misconceptions about programming languages. First of all, any language may be compiled or interpreted: these are properties of implementations, not languages.

Secondly, we have deliberately built two-stage language processors. The common front end inputs text and builds an AST, and the back end directly executes the AST, or generates code from it. But we haven’t specified where the input for the front end comes from. Either language processor might be called from the command line with a file argument, sit in a loop taking input directly from the keyboard, or be invoked within an IDE. Thus, a language is not in itself batch or interactive: these are properties of environments. 



■ FIG01: Interpreter and compiler phases

Greg Michaelson is interested in the design, implementation and analysis of programming languages, especially for multi-processors. He has taught programming for nearly 40 years.

CODE CLUB: DIGITAL MAKING

We talk to Code Club's Clare Sutcliffe to find out how far Code Club has come and what's in store for 2017



■ Code Clubs have proven to be massively popular in Australia, although clubs aren't always this big

The success of Code Club, a network of after-school clubs that help students learn computer science through fun activities, is due in no small part to the hard work of both volunteers and its co-founder, Clare Sutcliffe. There are over 5,000 Code Clubs in the UK alone, and more start every day. With over 4,000 Code Clubs outside the UK in 120 countries, Clare recently visited Hong Kong, Australia, and New Zealand to find out how clubs in other countries operate, and to see if any improvements were needed:

"I was looking for differences in the way Code Club was received by children in different countries," she tells us. "Generally, the clubs were very similar, which means that what we've created is scalable. There are subtle differences, but we work with particular partners to accommodate their needs in a sensitive way.

"We don't want to force people into impractical teaching situations."

With Australia changing their IT curriculum, and New Zealand introducing computing in 2018, Code Club is well-placed to handle the surge in interest. Local teams are crucial to enabling this:

■ Clare Sutcliffe is the co-founder of Code Club and Executive Director at Raspberry Pi



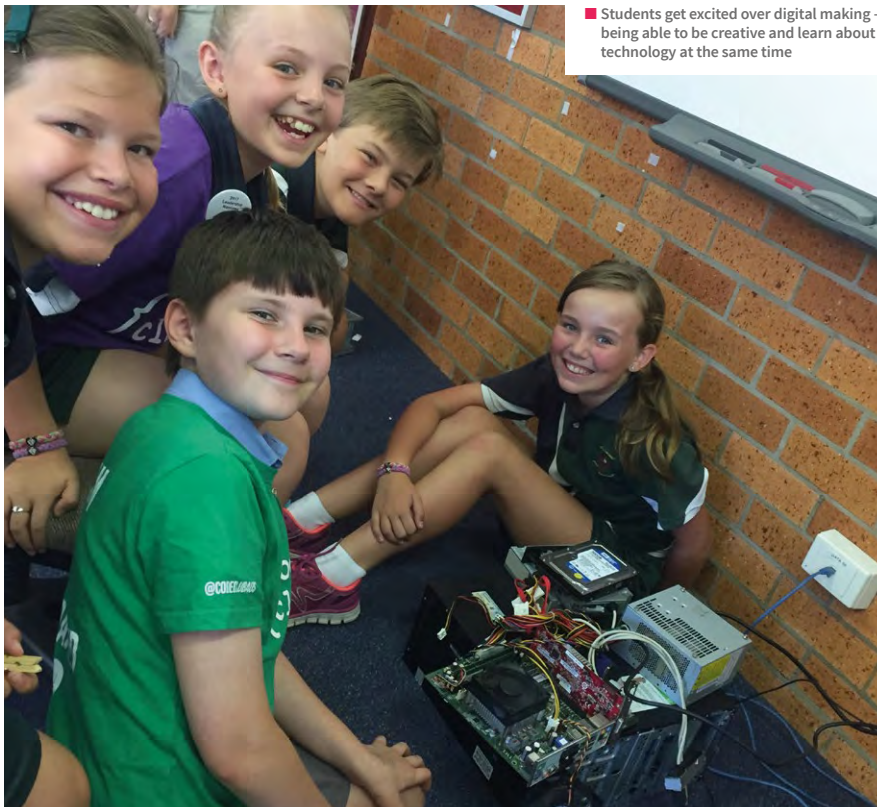
INTERNATIONAL ADVENTURES

Clare recently toured Code Clubs in other countries, to see their day-to-day operations:

"I visited Hong Kong, one of our most recent Code Club partners. There's only a small number of clubs there at the moment, but in 2017 they're planning to expand. We visited local schools on the large social estates, where they have very little in terms of resources. This means any volunteers are welcomed with open

arms. We saw the classroom setups and talked to the teachers about when they could start: we literally set them up while we were there! By Saturday morning, we had a little Code Club set up in a company office.

"No matter where you are in the world, the core of Code Club stays the same: children learning about technology by making things. That remains the same, regardless of what language you speak."



■ Students get excited over digital making – being able to be creative and learn about technology at the same time

“The Australian team travel the country training teachers, using a course similar to Code Club’s, then encourage them to set up their own club. It seems to be working well: one school had 90 children in one Code Club, and now run it three times a week!” Clare adds that many of the schools she visited were in underprivileged areas, which makes spreading the Code Club mission all the more important.

great to see volunteers from all around the world taking on our projects and starting a new Code Club.”

It’s easy to set up a Code Club account to get all the resources you need; all you need after that is a venue, and enthusiasm for teaching children about coding and digital making. Clare explains:

“Register your club on the website and we’ll send you new, exciting things that we’re

“ IF MATERIALS ARE AVAILABLE IN DIFFERENT LANGUAGES, WE CAN WIDEN ACCESS TO THEM

Looking ahead to 2017, Code Club’s priority will be worldwide expansion. “We’re going to be welcoming new partners,” Clare says. “We’re looking at countries interested in Code Club that we can have the most impact on. Translating material into a wide range of languages is important, especially the five most spoken languages in the world.

“We welcome help from volunteers here; if materials are available in different languages, we can widen access to them. It would be

working on. You can update your club details, and see which countries have a Code Club community up and running already. Discover what it takes to start a individual club at CodeClubworld.org.”

Ultimately, it seems that all it really takes is a little passion for the subject and a desire to teach the next generation vital digital skills, furnishing them with the tools they need for their future careers and opening up a whole new world of possibilities. (Hw)

CODE CLUB IN NUMBERS

5000+
CODE CLUBS
IN THE UK

4000+
CODE CLUBS
IN THE REST
OF THE WORLD

120+
COUNTRIES
CODE CLUBS CAN
BE FOUND IN

125,000
CHILDREN
IN CODE CLUBS
WORLDWIDE

20
LANGUAGES
PROJECTS TRANSLATED TO



■ There's no cost involved in starting a Code Club: it's free for schools and the kids who attend

STORY BY Emma Norton

START A CODE CLUB IN YOUR SCHOOL

Are you a teacher interested in setting up a Code Club in your school?
Find out how simple it is to get started...

TACKLING THE COMPUTING CURRICULUM

Running an after-school Code Club can help you to develop confidence to teach the computing curriculum, and to integrate computing into your everyday lessons.

If you and your colleagues are keen to get some additional, more formal training, you may be interested in Code Club's Teacher Training courses. There are three modules on offer, focusing on 'Computational Thinking', 'Programming and Networks', and 'The Internet'.

Many of the sessions are now free for teachers, so if you're interested, you can make an enquiry with the Code Club team by emailing hello@codeclubpro.org.

It's easier than you think to run a Code Club yourself: you don't need existing coding skills, just a can-do attitude to get stuck into learning alongside your students for an hour a week!

If you haven't heard about Code Club, it's a UK-based non-profit organisation offering free learning materials and support for teachers and volunteers running after-school coding clubs for children aged from nine to eleven.

Code Club's specially designed projects offer structured and fun content for the clubs. The projects are step-by-step guides for children to follow to create animations, games, websites, and much more. Children build up their programming skills as they move through the projects. There are also challenges to provide opportunities to apply what they've learnt.

Caroline Harding, a Year 4 teacher who helps to run a Code Club at her school in

Croydon, told us about the benefits the club has brought the children. "Making Code Club available to the children in our school has helped tremendously with their confidence and engagement in coding and computing in general," Caroline says. "It taps into their problem-solving skills and enables them to develop critical thinking skills. Programming and coding is an area of the curriculum that many staff can find intimidating. Knowing that the children have some experience of the program can help ease some anxieties and enables that 'have a go' attitude!"

By starting a club at your school you'll be joining a huge community of teachers who do the same thing: around 50% of Code Clubs are run by teachers.

If you're considering getting a Code Club started, we have come up with a few tips to help you.



■ Code Club allows children to experiment and invent, using different languages to create their own games, animations, and websites

Register your club online

To access Code Club's project materials, you will need to register online. You can sign up as a Code Club Host by visiting jump.to/cc/teachers, making sure to use your school email address so we can validate you as a teacher.

Once you've entered your details, you'll be able to select the option to run the club yourself. Your club will then be automatically activated and you'll have immediate access to all Code Club's online resources.

Code Club have projects in three different coding languages: Scratch, HTML/CSS, and Python. Beginning with Scratch is recommended, as this visual block-based language provides a great introduction to key programming concepts. If your pupils are already experienced with Scratch, though, you may wish to get started with HTML/CSS or Python. There are twelve Code Club projects in each language to keep your club occupied for a full term.

Your first Code Club

It's worth preparing for your first Code Club session by working through the project in advance, so that you're aware of all the instructions and the places where pupils could possibly get stuck.

Code Club is fun and it offers children (and their teachers) the opportunity to get creative with coding. It's a chance to experiment and invent, helping children to learn an important skill for their future, while engaging with technology and creating things that they can get excited about.

The model can be adapted to suit different venues and educational needs. Most clubs run through one coding project per week, but some children like to spend longer perfecting their designs. Some clubs have pairs of students sharing computers, and many clubs also like to experiment with physical computing. You can customise your club to suit you and the children.

Code Club in practice

There are thousands of teachers running their own Code Clubs across the country, and around the world. We spoke to

Matthew Cave, assistant head teacher at West Town Lane Academy in Bristol, who told us about his club.

Beginning with Year 5 and 6 students, Matthew and his team introduced Code Club's Scratch projects for all Key Stage 2 children. Now, they have a whole-school approach, with ScratchJr introduced for Key Stage 1, and they have invested in new technology including Lego WeDo and My Romo.

Matthew says, "We've been running our Code Club for over a year now, with 40 children attending. The club is in high demand."

Code Club's fun approach has provided other benefits: "It's amazing to see the sense of achievement the children get when they finish their projects. We can really see them starting to persevere with the tasks in Code Club, using analytical thinking to troubleshoot."

What advice does Matthew have for teachers who are thinking of starting a club? "It's dead easy, so take the plunge! The children will run with it, so don't worry about not being an expert." (HWT)



STORY BY Phil King

ROBOTS INVADE THE CLASSROOM

Find out how one teacher is using robotics to teach computing and other subjects to 11-12 year olds

A throng of miniature wheeled robots whizz around the classroom floor, much to the delight of the schoolchildren. Far from being irate, however, the teacher is delighted that her students are hard at work on their latest class project.

The school is Kings Glen Elementary in Springfield, Virginia, USA, where sixth-grade teacher Lisa Rode has introduced a group of GoPiGo robots, each with a Raspberry Pi single-board computer 'brain', to help teach computing and technology.

"I became interested in using Raspberry Pis in my classroom over the summer in 2014," she recalls. "I was intrigued by all of the possibilities that [it] provided for learning." This resulted in her applying successfully for a competitive grant to

start an after-school robotics club.

One of Lisa's main aims was to teach students how technology actually works, rather than just using it as a tool in the classroom. "Technology has transformed the way we teach, and students are taught skills to use it, but students seldom learn how it works," she explains. "Teaching students how to code and to problem-solve through robotics helps give [them] necessary skills for their future, as the number of computing-related employment opportunities continues to increase."

Far from already being an expert on the subject, Lisa had no prior experience working with electronics or robotics. "I've always enjoyed learning about and tinkering with new technologies, but I don't

have a formal background in it. Throughout this experience, I've learned about it alongside my students."

Thanks to ample support and resources



MARS ROVER PROJECT



Designed by Dexter Industries, the Mars Rover project (helloworld.cc/2jywL9V) is a group of lessons that teach students about past and current space technology. The students then apply that information to designing and building a planetary rover of their own, using the GoPiGo robot as a base.

"Students add real and fake sensors to their robot," reveals Lisa. "If students don't have access to any sensors, they will create models of them to attach to their robot." Students then send their robot onto a new 'planet' to explore and determine if the land is habitable for humans. "The 'planet' is a space that has been created to look like another world. In my class it was made out of cardboard, cups, foam blocks, and other materials from our class makerspace."

After an introductory lesson, the students gather data from their robot using an ultrasonic sensor and Raspberry Pi camera to see what the planet looks like. In the next lesson, they use a temperature and humidity sensor to gather more information about living conditions on the planet. They then compare the temperature of the planet to others in our solar system. Finally, students use a light sensor to gather data about day and night on the planet.

from GoPiGo manufacturer Dexter Industries, she found it easy to get started using the robots. "A group of my students volunteered to stay in class during lunch, and we learned about the robots and how to program them together. It's important to have good support and information to help teachers, especially if computing isn't part of their curriculum or background. The support from Dexter was vital, especially since this was a new experience for me."

■ Disguised as a crocodile, the GoPiGo robot heads for the shade when the sensed temperature is high, using data from a light sensor



Engaging students

There was an enthusiastic response from the schoolchildren when the robots were introduced into classroom lessons. "The students were surprised and more motivated to learn. When I applied for a contest grant to fund the robots, the students and whole school were very supportive and excited about the new opportunities." After an initial order of ten robots arrived, and the school saw the resulting enthusiasm of students,

facts about space exploration to being invested in an exploration of their own."

It's not just maths and science subjects that have benefited, as students have also used the robots during writing lessons. "For one lesson, students create a tour of a park

“TEACHING STUDENTS HOW TO CODE AND TO PROBLEM-SOLVE THROUGH ROBOTICS HELPS GIVE [THEM] NECESSARY SKILLS

Lisa was able to obtain extra funding for further resources.

Using the GoPiGo robots, her class has done a variety of activities in different lessons. "One of our major science units is astronomy. Through the Mars Rover project, students apply knowledge about instruments, equipment, and robots used in real space exploration to create their own, using the GoPiGo robot as a base. This allows students to go beyond just learning

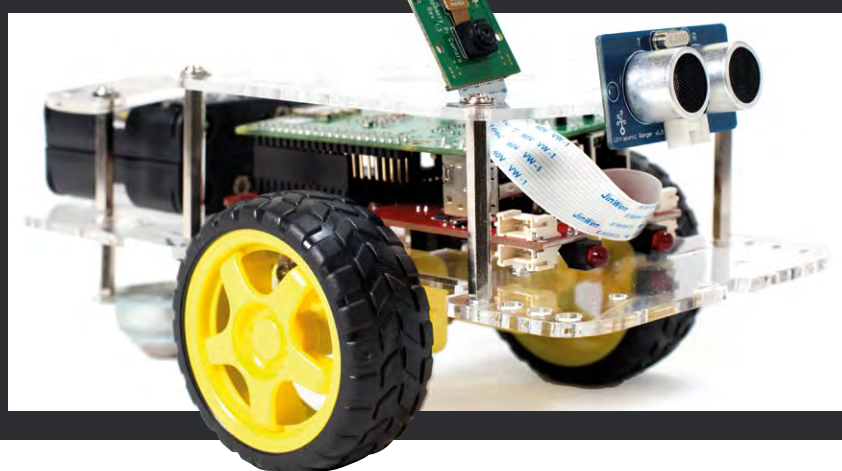
that they create. The GoPiGo then uses the line-follower sensor to travel through the park, and the tour is broadcast through a speaker on the robot. Some student projects were fictional parks, while others required research to gather information for the park. Students created zoos, tours of presidential homes, theme parks with imaginary creatures, and a park featuring major composers and artists."

The block-based, easy-to-learn Scratch programming language (scratch.mit.edu)

GOPIGO ROBOT

Launched via a successful Kickstarter crowdfunding campaign last year, the GoPiGo turns your Raspberry Pi into a fully functional two-wheeled robot (helloworld.cc/2iAbjgz). The \$100 Base Kit includes the GoPiGo board, chassis, wheels, motors, encoders, and battery pack. If you're starting from scratch, the Starter Kit adds a Raspberry Pi 3, mini WiFi dongle, GoPiGo servo package, ultrasonic sensor, micro SD card (with Dexter's software preloaded), power supply, and Ethernet cable. Classroom bundles and extra learning resources are also available.

According to its creator John Cole, of Dexter Industries, the GoPiGo was originally designed to make robotics accessible to everyone, especially in education. "We've got software for it in a lot of languages at this point, including Scratch and Python. Our hope was to give students a place to start with robotics, and leave the upside or potential as wide as possible."



► has been used by students to plan stories. "Students first write code for the GoPiGo robot, and then they create a storyline that reflects the events in the code. For example, the robot is programmed to move forward, stop and wait for a few seconds, and then move backward. The

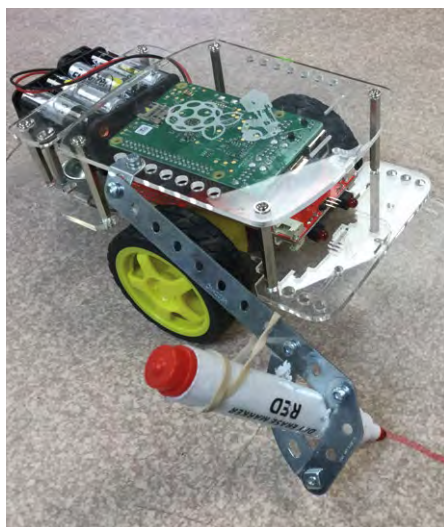
story could be that the main character is walking through the forest until suddenly they stop, because they see a bear staring right at them. They stop in their tracks and try to determine what to do next. Next, they slowly back away to try to get to safety."

Problem-solving

Lisa tells us that using the robots also helps to teach problem-solving skills during the after-school club activities. "One activity we did last year reflected the weather that had just occurred. Students had several days off school due to a blizzard. The first robotics club meeting after [it], students were given the task of creating a robot that could transport needed supplies to a family stranded due to a blizzard. The students created containers on their GoPiGo robot to hold supplies, and then programmed the robot to follow a certain path to get to the cabin to deliver the supplies and return home. Students had to work through several iterations of their design and programs before succeeding at their mission. This was a good way to teach not only programming, but the art of perseverance."

Another activity from the after-school programme is called Treasure Hunters and teaches students about the infrared line-follower sensor. "Students are trying to get their robot to the 'treasure' marked by an X on the floor. They're successful when the line-follower sensor reads all black and is completely over the X marking the location of the treasure. This was also a good activity for not only teaching how to program the robots, but problem-solving and using failure to learn and improve."

Students have also added arms or extensions to their robots to enable them to write and draw. "They were given the challenge to program their robot to create a picture or to write a letter or word. This



■ After-school club attendees added an arm to a robot, to enable it to draw pictures and write letters



was a good way for students to get used to using the robots and learn about the basics of coding.”

Lesson planning

In addition to the lessons and activities she has already done with her class, Lisa is excited about several lessons that she has developed for the robots and has not yet used in her classroom. “There are so many possibilities across the curriculum to integrate robotics into multiple content areas. One of the lessons I created is about cold-blooded animals. Students learn about the differences between cold-blooded and warm-blooded animals. They then transform

Another lesson sees students investigate how the GoPiGo robot works, and then compare and contrast the different human body systems to the robot’s components.

Maths lessons written by Lisa include investigating circumference and ratios. “In the circumference lesson, students learn about the connection between circumference and distance travelled, using the GoPiGo robot to measure how far it has travelled after one or more revolutions of the robot’s wheels.”

Lisa’s school curriculum focuses on the core content: maths, reading, writing, oral communication, science, and social studies. “Any programming and computing lessons



“ THE USE OF ROBOTS HAS ENGAGED STUDENTS THAT ARE NOT ALWAYS ENGAGED IN LESSONS.

their GoPiGo robot into a cold-blooded animal that reacts to changes in temperature. When it becomes too cold, it moves to a warmer spot and when it’s too hot, it moves to a cooler spot.”

should correlate to the core curriculum. This has made me look at different ways of planning and organising my lessons trying to incorporate physical computing whenever possible.”

Physical computing

As well as being a lot of fun, Lisa reckons that the use of physical computing (interactive physical systems that can sense and respond to the real world) in the classroom is helping her to teach the fundamentals of computing in a more engaging way. “When students see something in real life moving or reacting, they get much more excited than when their code only changes something virtually.”

It’s not just the more able students who find it helpful, either. Lisa has found that the use of robotics in the classroom benefits and appeals to children of all abilities. “The use of robots has engaged students that aren’t always engaged in lessons. They are allowed more freedom and creativity in their learning; it’s hands-on and not worksheet-based. The robots also give the students a safe space to fail and learn from failure. It’s not as intimidating to fix a problem with their robot as it is to struggle on a test or quiz. Students can then learn how to grow and learn from failure, rather than viewing it as an end point.”

The success of the robots in the classroom has led to more children attending the after-school robotics club. “The after-school club has grown since it first started. We currently have an eight-week session in the spring (once a week).”

In addition, other teachers have shown an interest in using robots in lessons. “There are a couple of other teachers at my school and surrounding schools starting to use robotics in class. They’re mainly used for extension activities.” (HAW)



■ The enthusiastic reaction from students has resulted in a greater motivation to learn about computing and technology

SARAH FILMAN & BROOK OSBORNE CODE.ORG

A COMPUTER SCIENCE COURSE FOR ALL HIGH-SCHOOL STUDENTS

Code.org's Computer Science Principles course provides a broad and approachable introduction to computer science for high-school students and teachers new to the field

Code.org is a non-profit organisation dedicated to expanding access to computer science, and increasing participation by women and under-represented minorities. Our vision is that every student in every school should have the opportunity to learn computer science, just like biology, chemistry, or algebra. To support this goal, Code.org has developed a pathway of courses and educational tools for students in primary and secondary school, including an Advanced Placement CS Principles course designed for students in grades 9-12 (years 10-12 in the UK).

A course for everyone

Code.org's CS Principles course is aligned to the Advanced Placement CS Principles Framework created by the College Board. It serves as an introduction to students new to computer science, and intentionally covers more than just traditional programming skills. Students in the course learn about the various protocols of the internet, how digital information is encoded, the security and privacy trade-offs of big data, and how to program interactive apps in JavaScript.

The curriculum is also intended to support teachers who are new to the field. Daily lesson plans, with activity guides and assessments, are provided to structure the year and ensure coverage of the CS Principles Framework. Activities put the student at the centre of the learning, so that the teacher doesn't need to be the single source of knowledge. Videos are also available throughout the curriculum, either providing direct guidance to teachers, or covering more complicated computer science concepts in a visual and engaging way.

Flexible programming paradigm

While the course is accessible to students with no experience, the curriculum and tools can support students with more experience as well. App Lab (helloworld.cc/2ijEfsh) is a programming environment developed by Code.org that supports transitioning from block-based programming to JavaScript text and back. This flexibility allows students working on the same projects to operate in the paradigm in which they are comfortable (blocks or text). The environment also supports many general JavaScript commands, so students who are able to take their creations further can do so.

Discovery through educational tools

To support meaningful understanding of the CS Principles learning objectives, Code.org has developed a set of educational tools designed to promote student exploration. These online sandboxes introduce rigorous, often abstract concepts in a concrete way, giving students a shared experience to reflect on as they move through the course. The Internet Simulator is a tool which gives students space to grapple with the design challenges of creating robust protocols to support communicating over the internet. Students have the freedom to build their own novel solutions, rather than being constrained to accepted solutions from the field. Other tools to support understanding of compression, pixels, and encryption, concepts that are often explored through paper-and-pencil activities, are integrated throughout the course. The Code.org CS Principles curriculum is totally free, open-source, and available online to any educator, regardless of background or experience. Learn more at code.org/cps. 



NICK PROVENZANO TEACHER

IMPOSSIBLE

Author, geek, and nerdy teacher, Nick Provenzano, on why failure matters...

Impossible. That was the first thought I had when I decided to jump into the world of computing and digital making. I'm a high-school literature and composition teacher, and it took a couple of students working on a passion project to pique my interest in coding. Once I saw "hello, world" appear on the screen, I was hooked. You too might be beginning your adventures into the world of computing and digital making, and I want to give you a few tips I've picked up on my journey.

A connected community

Embrace the maker community and use it when you're stuck. One of the best things I've experienced since I entered the world of digital making is the maker community. It's a diverse group of people that are smart, connected, and very helpful. Some of my first projects would not have been possible if it wasn't for the help from the community. If my code wasn't running, I couldn't get the light to blink, or a whole host of other issues occurred, they were solved with the support of the community. After a while, you'll find yourself helping others with their questions.

Take some fun risks when deciding on projects. The exciting part of trying something new is that you have no idea how it's going to end. Making is about taking risks and creating something fun. No idea is too crazy, and the biggest ideas can turn into the best learning experiences. I once turned a rotary phone into a working AirPlay system. It was a crazy idea that took some time to figure out, but

I did it. I didn't know how when I started, but now I do. Taking the risk allowed me to explore new ideas and learn great things. Try taking more risks, and you never know what cool stuff will happen.

Accept failure

Along with risk comes failure, and you need to accept it as part of the making process. There are going to be short-term failures and long-term failures. One project took me six months to get correct. It was a line of code I couldn't

“ One of the best things I've experienced since I entered the world of digital making is the maker community

figure out, but I eventually got it. When I did, I felt like a rock star! These failures teach valuable lessons for you as you advance on your computing journey. Also, don't be afraid to make your failures public. You'll get more help if you let people know that you need it, but it also shows your students that you're human and make mistakes as well.

I hope these few tips help you as you advance on your own making adventures, and I can't wait to share the fun and exciting things I discover over the coming months. **(HW)**

Nicholas Provenzano Four words describe Nicholas: Teacher.Learner.Learner.Nerd. He travels the country speaking and consulting with educators to share innovative practices.



STORY BY Matt Richardson

AMERICAN PICADEMY

By launching a new professional development opportunity for educators in the United States, the Raspberry Pi Foundation brings digital making stateside...

Training educators is one way that the Raspberry Pi Foundation achieves its mission of putting the power of digital making into the hands of people all over the world. Spreading this training beyond the borders of the United Kingdom, therefore, became a high priority in 2015.

JOIN US AT PICADEMY!

The Raspberry Pi Foundation is always on the lookout for engaged and enthusiastic educators to experience Picademy.

At helloworld.cc/2jqFVCd you can see the latest news on the United States program, and sign up for email alerts about this free two-day workshop for educators of all types.

At the same time, during the National Week of Making, President Obama made a call to action to create a nation of makers within the United States.

"During National Week of Making, we celebrate the tinkerers and dreamers whose talent and drive have brought new ideas to life, and we recommit to cultivating the next generation of problem-solvers," said President Obama. "As the maker movement grows, I continue to call on all Americans to help unlock the potential of our nation and ensure these opportunities reach all our young people, regardless of who they are or where they come from."

The President's call to action resonated with our commitment to digital making. At that time, the Raspberry Pi Foundation's Picademy programme had trained hundreds of teachers at venues across the

United Kingdom. These educators were getting hands-on training for teaching with digital making as a tool. Among those educators, some even travelled all the way from the United States to experience Picademy first-hand.

So it was clear to us at the Raspberry Pi Foundation that demand was strong within the United States for our special blend of digital making professional development. We therefore made a commitment in response to President Obama's call to action. In 2016, we promised to train at least 100 educators at four events on U.S. soil as a part of a U.S. pilot of Picademy. Making this commitment was the easy part. We knew that planning and executing the program would be a journey into uncharted territories.



PROJECT SPOTLIGHT



WHISKERS

Built with an Explorer HAT, motor, LEDs, and some arts and crafts supplies, Whiskers is a virtual coach programmed in Scratch to dispense helpful advice to teachers in need. It was created on the second day of Picademy Austin by Kimberly Boyce-Quentin and Bradley Quentin. You can think of it like a tongue-in-cheek Magic 8-Ball for teachers, dispensing advice such as “I’ve found that it’s better to ask for forgiveness instead of permission.”

Putting a plan into action

Picademy is an intense experience, no matter where it’s held. Over the course of the two days, all types of educators get hands-on experience with digital making. On the first day, they’re given a series of workshops that explore different ways to

■ Becoming a Raspberry Pi Certified Educator is just the beginning. (Image courtesy of Melissa Huch)

new program was a good opportunity to try things in a different way. Nothing was taken for granted. We scrutinised everything: the application process, content, agenda, equipment needs,

“ WE NEEDED TO MAKE SOME FAST DECISIONS ABOUT WHAT PICADEMY WOULD BE LIKE IN THE UNITED STATES

use Raspberry Pi in educational contexts. On the second day, they collaborate together to create something, using what they learned. The program culminates with the educators sharing their projects and what they learned with their peers.

As soon as our U.S. pilot got the green light, we needed to make some fast decisions about what Picademy would be like in the United States. We looked at every aspect of the program and examined its purpose. We had to evaluate whether it should be adapted in any way for American educators, or if this

and size of the cohort. Many changes were made, both big and small, but the essence of the program remained the same. These educators were going to learn a lot, fail a little, collaborate together, and have fun.

The collaboration between our Cambridge UK and San Francisco teams meant late evenings and early mornings for them, respectively. Working together over regular videoconferences, we discussed the what, when, why, and how. Figuring out the where was definitely the biggest challenge.

Location, location, location

Even today, our footprint in the United States is small. Hosting 40 educators for two days meant that in order to pull this off, we’d have to find a place to run the workshop. While it was a major challenge to find the right venues, it provided a great opportunity to enhance the program. What if, in addition to hosting the workshop, we also found institutions that could contribute to the content of Picademy? This would make the opportunity to learn with us even more valuable for the educators.

Because of this, the Computer History Museum in Mountain View, California was a natural partner from the beginning. In addition to providing the physical space to hold our first ever Picademy in the United States, they would also be able to take the attendees on a guided tour of their artifacts and facilitate a gallery discussion around the history of computing.

After the successful execution of the first two US Picademy workshops at ▶



FROM A PICADEMY FACILITATOR

Amanda Haughs, a Teacher on Special Assignment (TOSA) from Campbell Union School District in California, shares her experience as a Picademy participant and facilitator.

AS AN EDUCATOR, WHAT WAS THE BENEFIT OF PICADEMY?

Attending Picademy provided me with the knowledge that I needed to introduce more students and teachers in my school district to coding, computer science, and digital making.

WHAT'S IT LIKE BEING A PICADEMY FACILITATOR?

Having been given the opportunity to also facilitate at Picademy events has supported my continued growth as a digital maker. Each time that I work with a new cohort, I learn something new about not only the content, but also about the best ways to present the skills and concepts to both educators and the students of all ages with whom those educators work.

WHY IS THAT IMPORTANT?

The ongoing learning and sharing has given me the confidence and credibility I needed to begin advocating for a formal computer science pathway in our district, a project that I'm excited to say my instructional technology team has now seriously started discussing since my experiences with the Raspberry Pi Foundation!

► the Computer History Museum, we took the show on the road and found more institutions to host and contribute to the program. Our Austin cohort was treated to a tour of some of the most powerful supercomputers in the world, housed on-site at the Texas Advanced Computing Center. Our Baltimore host, the Digital Harbor Foundation, shared their experience of running an after-school technology centre.

"For us, Picademy was an opportunity to serve as a hub for passionate educators to come together and not only experience what the Raspberry Pi Foundation had put together, but to see what the Digital Harbor Foundation was about," says Shawn Grimes, DHF's Interim Executive Director. "In between the sessions, our staff were exchanging stories and ideas with participants to build upon each other's ideas."

All of our hosts provided much more than just a space to hold the workshop. They enhanced Picademy to make each session an incredibly unique experience for our educators.

From learning to leading

The 160 educators that joined us for the Picademy workshops in the United States were energetic, engaged, and from

many different educational contexts. They included classroom teachers, librarians, museum educators, and after-school educators. Even though they had varying levels of experience with the subject matter when they showed up, their first day equipped all of them with the basic knowledge that they'd need for employing digital making in their educational context.

"After the first day, I had a good sense of what was possible with the Raspberry Pi," says Jaymes Dec, a Technology Integrator at the Marymount School in New York City. "The next day, I was really excited to find myself in the same position that I often put my students: trying to find the answers to technical questions on my own before asking for help from the facilitators. It was a great opportunity to feel what they feel, as I discovered questions and then dug up the answers. In the end, I had a great time learning by making with my teammate."

Some of our newly inducted Raspberry Pi Certified Educators even come back to other workshops to facilitate the learning experience for others.

"A colossal value in coming back as a facilitator was being able to take what I learned during my session of Picademy, and turn it into a beneficial experience for other educators," says Venus Montes, a Computer

■ Picademy Certified Educator and facilitator Kevin Olson helps an educator. (Image courtesy of Melissa Huch)





■ Educators are encouraged to celebrate every success they have along their journey. (Image courtesy of Melissa Huch)



■ At the end of the second day, teams present their projects to their peers. (Image courtesy of Jorge Salazar)

Programming Instructor in North Bergen, New Jersey. “Being a facilitator extended the reach of what I learned during my Picademy session beyond my classroom, my district, and my state, into classrooms of many other educators across the nation.”

the #picademy hashtag is always full of great ideas and troubleshooting, and the personal connections I’ve made through Picademy continue to enrich my personal and professional life.”

Getting this digital making community started among educators in the United

“ I NOW HAVE A WHOLE NETWORK OF COLLEAGUES I CAN TURN TO FOR SUPPORT AND TO SHARE IDEAS

Bootstrapping a community

But the Picademy experience doesn’t end when the two days are over. It’s an induction into a very large community of Raspberry Pi Certified Educators worldwide. These educators help each other, share their students’ work, meet up at Raspberry Jams, and train other educators.

“Picademy has opened up so many doors to me. I now have a whole network of colleagues I can turn to for support and to share ideas,” says Certified Educator Kevin Olson. “The Google+ community is a fantastic resource of like-minded individuals,

States is an important part of Picademy. Having a strong community of educators is critical to achieving our mission. The Picademy program is, by design, meant to get this community started.

“Being at Picademy helped introduce me to a supremely supportive community,” says Matthew Buckley, a Director of Instructional Technology at Bishop McNamara High School, Forestville, Maryland. “The information we receive on day one is golden, but what was more helpful for me was being introduced and encouraged within the digital making community.”

Looking ahead

The impact of Picademy is measured not only by the number of teachers that we train, but also the number of students that each of them reaches.

“Possibly the single greatest outcome from Picademy is the confidence it has given me to jump in and get going with digital making and computer science,” says Kevin Olson. “I have 72 sixth-graders programming robots to recreate novels, two middle-school electives full of students who can’t wait to get their hands dirty with Sonic Pi and Minecraft, and my mind is constantly buzzing with ideas for integrating projects in all subject areas. And instead of waiting around to figure out every last detail, we’re jumping in and learning together!”

Thanks to a very successful pilot, the Raspberry Pi Foundation trained 160 educators in four Picademy workshops in Mountain View, Austin, and Baltimore. To build on this success, our goal is to train at least 300 educators in the United States in 2017 and bring the program to new regions. We’re always on the lookout for enthusiastic educators, no matter what their level of experience, to take part in this unique programme. Best of all, this professional development offering is completely free for educators.

We hope to welcome you to a Picademy soon. (HW)



MILES BERRY PRINCIPAL LECTURER

VALUES?

What's the place of ethics in our work as computing or digital making educators?
How can we help our students to help others?

A few things have left me pondering the place of values and ethics in computing education and digital making.

The Children's Commissioner recently published a report on children's online rights. I was asked, when presenting on the English national curriculum in Hanoi, why we didn't mention values, as they had in the framework they're currently drafting.

Back home, the European Parliament produced a draft report on the ethical principles that should underpin the development and design of robots.

As a community of digital educators, what are our shared values? What are the overarching aims or principles of what we're trying to do in computing education or digital making?

The English computing curriculum starts with this ambitious vision:

A high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world.

Understanding the world is an enlightenment value: it assumes the world is a knowable thing, and that curiosity about it is good. We might, then, see the need for free access to knowledge, free participation in debate, and freedom to experiment as part of the learning process. 'Changing the world' is ambitious, but I worry that we leave implicit the idea of changing it **for the better**. The English computing curriculum emphasises the need for pupils to stay safe and act responsibly, but shouldn't we also consider the ethical use of technologies to improve the lives of others, and the ethical assumptions of the algorithms behind the services we rely on?

The new US K12 CS Framework goes further, emphasising an inclusive culture as a guiding principle and the need to teach the impacts of computing:

An informed and responsible person should understand the social implications of the digital world, including equity and access to computing.

The Raspberry Pi Foundation has a clear mission statement:

To put the power of digital making into the hands of people all over the world, so they are capable of understanding and shaping our increasingly digital world, able to solve the problems that matter to them, and equipped for the jobs of the future.

It's impressive to see the founders' values in firstly making low-cost, general-purpose computers available to all and, secondly, prioritising education as a shared goal. Other similar projects share a sense of positive change through technology, like Apps for Good. Those involved learn skills and develop understanding, but they also make apps that have a societal benefit: a moral purpose.

I suspect that character, values, and ethics in education are better learnt through example than worksheet; We must bring these more to the surface in what we do: thinking about the why, as well as the what and the how of the things our students learn and make. **(HW)**

Miles Berry is principal lecturer in computing education at the University of Roehampton. He helped draft the English computing curriculum and is a member of the Raspberry Pi Foundation.



JOIN THE GLOBAL CODERDOJO COMMUNITY

Free, global, volunteer-led, coding clubs for young people aged 7-17. Code apps, build websites, make games, hack hardware and enjoy exploring technology together.



START A DOJO

VOLUNTEER AT A DOJO

SUPPORT THE MOVEMENT

RESOURCES & SUPPORTS

There is a host of tools & supports to enable you to empower youths:



**ONLINE
TRAINING**



**LEARNING
RESOURCES**



**COMMUNITY
SUPPORT**



**DIGITAL
BADGES**

See how you can join the community at coderdojo.com or get in touch with us at info@coderdojo.com



helloworld.cc